

普通高等教育“十一五”国家级规划教材



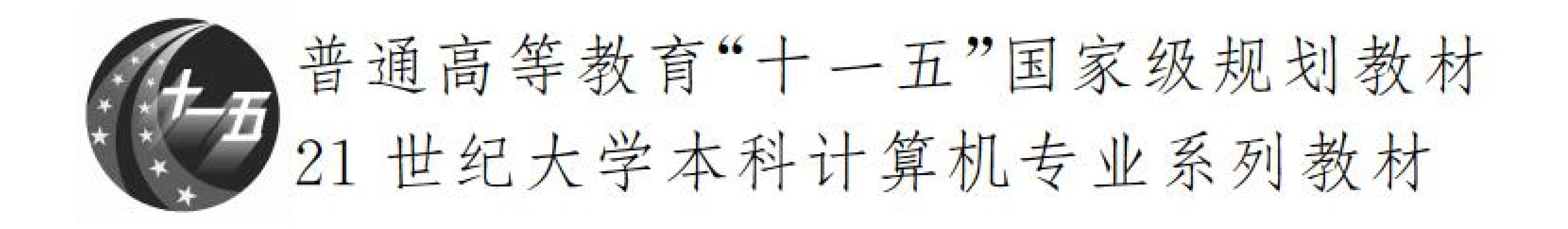
21世纪大学本科计算机专业系列教材

屈婉玲 刘 田 张立昂 王捍贫 编著

算法设计与分析习题解答与学习指导（第2版）

- 根据教育部“高等学校计算机科学与技术专业规范”组织编写
- 与美国 ACM 和 IEEE CS *Computing Curricula* 最新进展同步

清华大学出版社



普通高等教育"十一五"国家级规划教材
21世纪大学本科计算机专业系列教材

算法设计与分析习题解答与学习指导(第2版)

屈婉玲　刘　田　张立昂　王捍贫　编著

清 华 大 学 出 版 社
北 京

内容简介

本教材为普通高等教育"十一五"国家级规划教材《算法设计与分析(第2版)》(主教材)的辅助教材.主教材的主要内容包括基础知识、分治策略、动态规划、贪心法、回溯与分支限界、线性规划、网络流算法、算法分析与问题的计算复杂度、NP完全性、近似算法、随机算法、处理难解问题的策略等.本书对主教材所阐述的算法设计技术和分析方法进行了总结,并对其中200多道习题给出了详尽的解答和分析.

本书适合作为大学计算机科学与技术、软件工程、信息安全、信息与计算科学等专业本科生和研究生的辅助教学用书,也可以作为从事实际问题求解的算法设计与分析工作的参考书.

图书在版编目(CIP)数据

算法设计与分析习题解答与学习指导/屈婉玲等编著. --2版. --北京:清华大学出版社,2016
21世纪大学本科计算机专业系列教材
ISBN 978-7-302-42955-5

Ⅰ. ①算…　Ⅱ. ①屈…　Ⅲ. ①电子计算机—算法设计—高等学校—教学参考资料 ②电子计算机—算法分析—高等学校—教学参考资料　Ⅳ. ①TP301.6

中国版本图书馆CIP数据核字(2016)第024865号

责任编辑:张瑞庆
封面设计:何凤霞
责任校对:焦丽丽
责任印制:刘海龙

出版发行:清华大学出版社
网　　址:http://www.tup.com.cn, http://www.wqbook.com
地　　址:北京清华大学学研大厦A座　　**邮　　编**:100084
社 总 机:010-62770175　　**邮　　购**:010-62786544
投稿与读者服务:010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈:010-62772015, zhiliang@tup.tsinghua.edu.cn
课件下载:http://www.tup.com.cn,010-62795954
印 刷 者:北京富博印刷有限公司
装 订 者:北京市密云县京文制本装订厂
经　　销:全国新华书店
开　　本:185mm×260mm　　**印　　张**:12　　**字　　数**:286千字
版　　次:2014年8月第1版　2016年3月第2版　　**印　　次**:2016年3月第1次印刷
印　　数:3501～5500
定　　价:29.00元

产品编号:068216-01

第2版前言

FOREWORD

本书是《算法设计与分析(第2版)》的辅助教材.作为普通高等教育"十一五"国家级规划教材,《算法设计与分析(第1版)》出版已经近5年了. 在这5年的时间里,大数据、云计算、互联网+等新领域、新问题、新应用层出不穷,许多问题求解都离不开问题的建模和算法的设计与分析.这次关于主教材的修订保持了原书的基本结构、主要内容与写作特色. 考虑到线性规划与网络流问题在实践中的广泛应用,增加了这两章内容,并在第9章增添了整数线性规划的NP完全性证明. 此外,补充了部分习题,并对第1版书中的某些疏漏之处进行了更新.

与主教材配套,本书也进行了同步更新,增加了第6章线性规划、第7章网络流算法,并对补充习题给出了解答.

本书第1～4章由屈婉玲编写,第5、8章由王捍贫编写,第6～7、9～10章由张立昂编写,第11～12章由刘田编写.

欢迎广大读者批评指正!

作　者

2015年11月于北京大学

第1版前言

FOREWORD

作为问题求解和程序设计的重要基础，算法设计与分析在计算机科学与技术专业的课程体系中是一门重要的必修课．通过该课程的学习，不但为学习其他专业课程奠定了扎实的基础，而且对培养学生分析与解决问题的能力及计算思维有着不可替代的作用．*ACM IEEE Computing Curricula* 2004 与我国教育部计算机科学与技术专业教学指导委员会提出的《计算机科学与技术专业规范 2005》都把该课程列入本专业的核心课程之一．

本书是国家高等教育"十一五"规划教材《算法设计与分析》(清华大学出版社出版，屈婉玲等编著)的辅助教材．主教材包括算法设计、算法分析、计算复杂性理论等重要内容．结合各种典型应用，主教材首先深入分析了各种算法设计技术的适用范围、设计步骤、正确性证明与复杂度的分析方法、改进算法的途径、局限性等，为从事实际问题求解的算法设计与分析工作在理论上提供清晰的、整体的思路和方法，并在此基础上介绍了问题难度的分析方法和计算复杂性理论的基本框架和一些重要的结果．

算法具有广泛的应用背景，习题量大，方法灵活．针对给定算法问题，在建模、设计技术选择、效率分析、改进途径等方面，初学者往往不知道如何着手．本书在多年算法教学的基础上精选了 100 多道典型的习题，给出了详尽的解答和分析，以期对初学者有所帮助．

与主教材配套，本书也分为 10 章．第 1 章是基础知识；第 2～5 章分别阐述分治策略、动态规划、贪心法、回溯与分支限界等算法设计技术；第 6 章介绍算法分析和问题的计算复杂度；第 7 章是 NP 完全性理论；第 8 章是近似算法；第 9 章是随机算法；第 10 章介绍处理难解问题的策略．每章首先对所涉及的重要知识点和方法进行总结，然后给出习题和解答．

本书前 4 章由屈婉玲编写，第 5～6 章由王捍贫编写，第 7～8 章由张立昂编写，第 9～10 章由刘田编写．

为了提高本书的质量，欢迎广大读者的批评和指正！

作　者

2014 年 3 月于北京大学

目　　录

CONTENTS

第 1 章 基础知识

1.1 内容提要

1. 基本概念

算法 有限条指令的序列,确定了解决某个问题的运算或操作顺序.

算法 A 解问题 *P* 把问题 P 的任何实例作为算法 A 的输入,A 能够在有限步停机,并输出该实例的正确的解.

算法的时间复杂度

最坏情况下的时间复杂度:算法求解输入规模为 n 的实例所需的最多基本运算次数,通常记作 $W(n)$.

平均情况下的时间复杂度:针对输入规模为 n 的实例的概率分布,算法求解这些实例所需的基本运算次数的概率平均,通常记作 $A(n)$.

函数的渐近的界 设 f 和 g 是定义域为自然数集 $\mathbf{N}$ 上的函数.

(1) 若存在正数 c 和 n_0,使得对一切 $n \geqslant n_0$,有 $0 \leqslant f(n) \leqslant cg(n)$ 成立,则称 $f(n)$ 的**渐近的上界**是 $g(n)$,记作 $f(n)=O(g(n))$.

(2) 若存在正数 c 和 n_0,使得对一切 $n \geqslant n_0$,有 $0 \leqslant cg(n) \leqslant f(n)$ 成立,则称 $f(n)$ 的**渐近的下界**是 $g(n)$,记作 $f(n)=\Omega(g(n))$.

(3) 若对于任意正数 c 都存在 n_0,使得当 $n \geqslant n_0$ 时有 $0 \leqslant f(n) < cg(n)$ 成立,则记作 $f(n)=o(g(n))$.

(4) 若对于任意正数 c 都存在 n_0,使得当 $n \geqslant n_0$ 时有 $0 \leqslant cg(n) < f(n)$ 成立,则记作 $f(n)=\omega(g(n))$.

(5) 若 $f(n)=O(g(n))$ 且 $f(n)=\Omega(g(n))$,则记作 $f(n)=\Theta(g(n))$.

2. 某些重要的结果

定理 1.1 设 f 和 g 是定义域为自然数集合的函数.

(1) 如果 $\lim\limits_{n\to\infty}\dfrac{f(n)}{g(n)}$ 存在,并且等于某个常数 $c>0$,那么 $f(n)=\Theta(g(n))$.

(2) 如果 $\lim\limits_{n\to\infty}\dfrac{f(n)}{g(n)}=0$,那么 $f(n)=o(g(n))$.

(3) 如果 $\lim\limits_{n\to\infty}\dfrac{f(n)}{g(n)}=+\infty$,那么 $f(n)=\omega(g(n))$.

定理 1.2 设 f、g 和 h 是定义域为自然数集合的函数:

(1) 如果 $f=O(g)$ 且 $g=O(h)$,那么 $f=O(h)$.

(2) 如果 $f=\Omega(g)$ 且 $g=\Omega(h)$，那么 $f=\Omega(h)$.

(3) 如果 $f=\Theta(g)$ 且 $g=\Theta(h)$，那么 $f=\Theta(h)$.

定理 1.3 假设 f 和 g 是定义域为自然数集合的函数，若对某个其他的函数 h，有 $f=O(h)$ 和 $g=O(h)$，那么 $f+g=O(h)$.

关于几类函数的阶的结果

多项式函数：$f(n)=a_0+a_1n+a_2n^2+\cdots+a_dn^d$，有 $f(n)=\Theta(n^d)$.

对数函数：对每个 $b>1$ 和每个 $a>0$，有 $\log_b n=o(n^a)$；对于不同的 a 与 b，$a,b>1$，$\log_a n=\Theta(\log_b n)$.

指数函数：对每个 $r>1$ 和每个 $d>0$，r^n 满足 $n^d=o(r^n)$.

阶乘函数：斯特灵(Stirling)公式 $n!=\sqrt{2\pi n}\left(\frac{n}{\mathrm{e}}\right)^n\left(1+\Theta\left(\frac{1}{n}\right)\right)$.

求和方法 利用公式或者以积分作为近似结果，其中常用的公式是：

$$\sum_{k=1}^{n}a_k=\frac{n(a_1+a_n)}{2}$$

$$\sum_{k=0}^{n}aq^k=\frac{a(1-q^{n+1})}{1-q}$$

$$\sum_{k=0}^{n}x^k=\frac{1-x^{n+1}}{1-x}$$

$$\sum_{k=1}^{n}\frac{1}{k}=\ln n+O(1)$$

递推方程求解方法 公式法、迭代归纳法、尝试法、递归树、主定理.

主定理(Master Theorem) 设 $a\geqslant1,b>1$ 为常数，$f(n)$为函数，$T(n)$为非负整数，且

$$T(n)=aT(n/b)+f(n)$$

则有以下结果：

(1) 若 $f(n)=O(n^{\log_b a-\varepsilon})$，$\varepsilon>0$，那么 $T(n)=\Theta(n^{\log_b a})$.

(2) 若 $f(n)=\Theta(n^{\log_b a})$，那么 $T(n)=\Theta(n^{\log_b a}\log n)$.

(3) 若 $f(n)=\Omega(n^{\log_b a+\varepsilon})$，$\varepsilon>0$，且对于某个常数 $c<1$ 和所有充分大的 n，有 $af(n/b)\leqslant cf(n)$，那么 $T(n)=\Theta(f(n))$.

3. 算法

顺序搜索算法 Search(L,x)，在 L 中查找 x. 基本运算为 x 与 L 中元素的比较，算法最坏情况下的时间为 $W(n)=O(n)$.

插入排序算法 InserSort(A,n)，对 n 个元素的数组 A 排序，基本运算为 A 中元素的比较，算法最坏情况下的时间为 $W(n)=O(n^2)$.

二分归并排序算法 MergeSort(A,p,r) 对数组 $A[p..r]$排序，基本运算为 A 中元素的比较，最坏情况下的时间为 $W(n)=O(n\log n)$.

Hanoi 塔的递归算法 Hanoi(A,C,n)，把 n 个圆盘从 A 柱移到 C 柱，基本运算为 1 个圆盘的 1 次移动，时间复杂度为 2^n-1.

1.2 习　　题

1.1 设 A 是 n 个不等的数的数组，$n>2$. 以比较作为基本运算，试给出一个 $O(1)$ 时间的算法，找出 A 中一个既不是最大也不是最小的数. 写出算法的伪码，说明该算法在最坏情况下执行的比较次数.

1.2 考虑下述选择排序算法：

算法 ModSelectSort

输入：n 个不等的整数的数组 $A[1..n]$

输出：按递增次序排序的 A

```
for i←1 to n−1 do
    for j←i+1 to n do
        if A[j]<A[i] then A[i]↔A[j]
```

问：

(1) 最坏情况下该算法做多少次比较运算？

(2) 最坏情况下该算法做多少次交换运算？这种情况在什么输入条件下发生？

1.3 给定正整数的数组 $A[1..n]$，测试 A 的每个元素 $A[i]$ 的奇偶性. 如果 $A[i]$ 是奇数，则将它 2 倍后输出；否则直接输出 $A[i]$.

(1) 以乘法作为基本运算，使用大 O 记号，还是使用大 Θ 记号，哪个记号能够正确表达这个算法对于规模为 n 的输入所做的基本运算次数？为什么？

(2) 如果以元素的测试作为基本运算，重复问题(1).

1.4 计算下述算法所执行的加法次数.

算法 1

输入：$n=2^t$，t 为正整数

输出：k

```
k←0
while n≥1 do
    for j←1 to n do
        k←k+1
    n←n/2
return k
```

1.5 计数算法 C 所执行的加法次数.

算法 C

输入：n 为正整数

输出：k

```
k←0
for i←1 to n do
    m←n/i
    for j←1 to m do
        k←k+1
```

```
return k
```

1.6 阅读关于下述算法A的伪码,说明该算法求解的是什么问题,并计算该算法所做的乘法运算(*)和加法运算次数.

算法A

输入:数组 $P[0..n]$,实数 x

输出:y

```
y←P[0]; power←1
for i←1 to n do
    power←power*x
    y←y+P[i]*power
return y
```

1.7 下述Find-Second-Min算法是找第二小算法. 输入是 n 个不等的数构成的数组 S,输出是第二小的数 $SecondMin$.

(1) 在最坏情况下,该算法做多少次比较?

(2) 若所有输入是等概率分布的,平均情况下该算法做多少次比较?

算法 Find-Second-Min(S,n)

```
if S[1]<S[2]
then min←S[1]; SecondMin←S[2]
else min←S[2]; SecondMin←S[1]
for i←3 to n do
    if S[i]<SecondMin
    then if S[i]<min
        then SecondMin←min; min←S[i]
        else SecondMin←S[i]
```

1.8 已知 L 是含有 n 个元素并且排好序的数组,x 在 L 中. 如果 x 出现在 L 中第 i 个($i=2,3,\cdots,n$)位置的概率是在前一个位置概率的一半,当 n 充分大时,估计下述查找算法平均情况下的时间复杂度 $A(n)$. 只需给出近似值.

算法:顺序查找

```
j←1
while j≤n and x>L[j] do
    j←j+1
if x<L[j] or j>n
then j←0
```

1.9 设 A 为 n 个不等的数的数组. 给定 x,若 x 在 A 中,输出 x 的下标 k;若 x 不在 A 中,输出0. BinarySearch 和 LinearSearch 分别表示二分和顺序搜索,设计下述算法:

算法 Search(A,x)

1. if n 为奇数 then k←BinarySearch(A,x)
2. else k←LinearSearch(A,x)

以比较作基本运算,用大 O 记号表示算法在最坏情况下的时间复杂度 $W(n)$;能否使用大 Θ 记号表示 $W(n)$? 为什么?

1.10 考虑下述素数测试算法：

```
算法 PrimalityTest(n)
输入：n,n 为大于 2 的奇整数
输出：true 或者 false
s←√n
for j←2 to s
    if j 整除 n
    then return false
return true
```

(1) 假设计算$\sqrt{n}$可以在$O(1)$时间完成，估计该算法在最坏情况下的时间复杂度.

(2) 能否使用Θ符号表示这个算法在最坏情况下的时间复杂度？为什么？

1.11 证明定理 1.3：假设 f 和 g 是定义域为自然数集合的函数，若对某个其他函数 h 有 $f=O(h)$和 $g=O(h)$成立，那么 $f+g=O(h)$.

1.12 证明定理 1.4：对每个 $b>1$ 和每个 $a>0$，有 $\log_b n=o(n^a)$.

1.13 证明定理 1.5：对每个 $r>1$ 和每个 $d>0$，有 $n^d=o(r^n)$.

1.14 设 x 为实数，n、a 和 b 为整数，证明下述性质：

(1) $x-1<x\leqslant x\leqslant x<x+1$

(2) $x+n=x+n$，$x+n=x+n$

(3) $\left\lceil \frac{n}{2} \right\rceil+\left\lfloor \frac{n}{2} \right\rfloor=n$

(4) $\left\lceil \frac{\left\lceil \frac{n}{a} \right\rceil}{b} \right\rceil=\left\lceil \frac{n}{ab} \right\rceil$，$\left\lfloor \frac{\left\lfloor \frac{n}{a} \right\rfloor}{b} \right\rfloor=\left\lfloor \frac{n}{ab} \right\rfloor$

1.15 考虑下面每对函数 $f(n)$和 $g(n)$，如果它们的阶相等，则使用Θ记号；否则使用 O 记号表示它们的关系.

(1) $f(n)=(n^2-n)/2$，$g(n)=6n$

(2) $f(n)=n+2\sqrt{n}$，$g(n)=n^2$

(3) $f(n)=n+n\log n$，$g(n)=n\sqrt{n}$

(4) $f(n)=2\log^2 n$，$g(n)=\log n+1$

(5) $f(n)=\log(n!)$，$g(n)=n^{1.05}$

1.16 在表 1.1 中填入 true 或 false.

表 1.1 函数 f 与 g

	$f(n)$	$g(n)$	$f(n)=O(g(n))$	$f(n)=\Omega(g(n))$	$f(n)=\Theta(g(n))$
1	$2n^3+3n$	$100n^2+2n+100$			
2	$50n+\log n$	$10n+\log\log n$			
3	$50n\log n$	$10n\log\log n$			
4	$\log n$	$\log^2 n$			
5	$n!$	5^n			

1.17 对于下面每个函数 $f(n)$,用 Θ 符号表示成 $f(n)=\Theta(g(n))$的形式,其中 $g(n)$要尽可能简洁. 比如 $f(n)=n^2+2n+3$ 可以写成 $f(n)=\Theta(n^2)$. 然后,按照阶递增的顺序将这些函数进行排列.

$$(n-2)!,\quad 5\log(n+100)^{10},\quad 2^{2n},\quad 0.001n^4+3n^3+1$$

$$(\ln n)^2,\quad \sqrt[3]{n}+\log n,\quad 3^n,\quad \log(n!),\quad \log(n^{n+1}),\quad 1+\frac{1}{2}+\cdots+\frac{1}{n}$$

1.18 对以下函数,按照它们的阶从高到低排列;如果 $f(n)$与 $g(n)$的阶相等,表示为 $f(n)=\Theta(g(n))$.

$$2^{\sqrt{2\log n}},\quad n\log n,\quad \sum_{k=1}^{n}\frac{1}{k},\quad n2^n,\quad (\log n)^{\log n},\quad 2^{2n},\quad 2^{\log\sqrt{n}}$$

$$n^3,\quad \log(n!),\quad \log n,\quad \log\log n,\quad n^{\log\log n},\quad n!,\quad n,\quad \log 10^n$$

1.19 求解以下递推方程:

(1) $\begin{cases} T(n)=T(n-1)+n^2 \\ T(1)=1 \end{cases}$

(2) $\begin{cases} T(n)=9T(n/3)+n \\ T(1)=1 \end{cases}$

(3) $\begin{cases} T(n)=T\left(\frac{n}{2}\right)+T\left(\frac{n}{4}\right)+cn,\ c \text{ 为常数} \\ T(1)=1 \end{cases}$

(4) $\begin{cases} T(n)=T(n-1)+\log 3^n \\ T(1)=1 \end{cases}$

(5) $\begin{cases} T(n)=5T(n/2)+(n\log n)^2 \\ T(1)=1 \end{cases}$

(6) $\begin{cases} T(n)=2T\left(\frac{n}{2}\right)+n^2\log n \\ T(1)=1 \end{cases}$

(7) $\begin{cases} T(n)=T(n-1)+\frac{1}{n} \\ T(1)=1 \end{cases}$

(8) $T(n)=T(n-1)+\log n$,估计 $T(n)$的阶

1.20 设递推方程 $T(n)=7T(n/2)+n^2$ 给出了算法 A 在最坏情况下的时间复杂度函数,算法 B 在最坏情况下的时间复杂度函数 $W(n)$ 满足递推方程 $W(n)=aW(n/4)+n^2$. 试确定最大的正整数 a 使得 $W(n)$的阶低于 $T(n)$的阶.

1.21 设原问题的规模是 n,从下述 3 个算法中选择一个最坏情况下时间复杂度最低的算法,简要说明你的理由.

算法 A:将原问题划分规模减半的 5 个子问题,递归求解每个子问题,然后在线性时间将子问题的解合并得到原问题的解.

算法 B:先递归求解 2 个规模为 $n-1$ 的子问题,然后在常量时间内将子问题的解合并.

算法 C:将原问题划分规模为 $n/3$ 的 9 个子问题,递归求解每个子问题,然后在 $O(n^3)$ 时间将子问题的解合并得到原问题的解.

1.3 习题解答与分析

1.1 算法

输入：数组 A

输出：A 中既不是最大也不是最小的一个数

```
1.  任选 3 个数 a1, a2, a3
2.  if a1 < a2
3.  then if a1 > a3
4.         then return a1
5.         else return min{a2, a3}
6.  else if a2 > a3 then return a2
7.         else return min{a1, a3}
```

算法至多比较 3 次.

1.2 （1）最坏情况下的比较次数是：

$$W(n)=\sum_{i=1}^{n-1}(n-i)=1+2+\cdots+n-1=n(n-1)/2$$

（2）当输入的 n 个数彼此不等且按递降次序排列时，比较次数是 $n(n-1)/2$. 每次比较后都发生交换，因此最坏情况下的交换次数也是 $n(n-1)/2$.

1.3 （1）对于任意大的 n，若 n 个数都是奇数，则算法做 n 次乘法；若 n 个数都是偶数，则算法做 0 次乘法. 因此乘法次数可以表示为 $O(n)$，但是不能表示成 $\Theta(n)$.

（2）对于 A 中的每个数，不管是奇数还是偶数，算法都做 1 次测试，测试次数可以表示为 $O(n)$，也可以表示为 $\Theta(n)$.

1.4 第一次 for 循环执行 n 次加法，第 2 次 for 循环执行 $n/2$ 次加法……直到最后执行 1 次加法，加法总次数为

$$T(n)=n+\frac{n}{2}+\frac{n}{2^2}+\frac{n}{2^3}+\cdots+2+1=2n-1$$

1.5 该算法的 for 循环执行 n 次，总计加法次数为

$$W(n)=n+n/2+n/3+\cdots+n/n$$

由于

$$\sum_{i=1}^{n}\left(\frac{n}{i}-1\right)\leqslant\sum_{i=1}^{n}\left\lfloor\frac{n}{i}\right\rfloor\leqslant\sum_{i=1}^{n}\frac{n}{i}$$

于是

$$W(n)=\Theta(n\log n)$$

1.6 给定实数 x，算法 A 是对多项式 $P(x)=p_0+p_1x+\cdots+p_nx^n$ 求值，其中 $p_i=P[i]$，$i=0,1,\cdots,n$. 算法 A 执行 $2n$ 次乘法，n 次加法.

1.7 （1）行 4 的 for 循环执行 $n-2$ 次，每次至多做 2 次比较（行 5 和行 6）. 行 1 做 1 次比较，算法比较次数至多为

$$W(n)=2(n-2)+1=2n-3$$

(2) 对于 i,在 $S[1..i]$的数之间,如果 $S[i]$是最小的 2 个数,则需要比较 2 次;如果是较大的 $i-2$ 个数,则需要比较 1 次. 由于等概率,处于较小的 2 个数的概率是 $2/i$,处于较大的 $i-2$ 个数的概率是$(i-2)/i$,因此有

$$\begin{aligned} A(n) &= 1 + \sum_{i=3}^{n}\left(2 \cdot \frac{2}{i} + 1 \cdot \frac{i-2}{i}\right) \\ &= n - 1 + 2\sum_{i=3}^{n}\frac{1}{i} \\ &\approx n - 1 + 2(\ln n - 3/2) \\ &= n - 4 + 2\ln n \end{aligned}$$

1.8 设 x 恰好在第一位的概率为 p,在第二位的概率为 $p/2$……那么 x 恰好在第 i 位的概率为 $p/2^{i-1}$,于是

$$p + \frac{p}{2} + \frac{p}{2^2} + \cdots + \frac{p}{2^{n-1}} = 1 \Rightarrow p = \frac{2^{n-1}}{2^n - 1} \approx \frac{1}{2}$$

平均比较次数是

$$\begin{aligned} A(n) &= p \cdot 1 + \frac{p}{2} \cdot 2 + \frac{p}{2^2} \cdot 3 + \cdots + \frac{p}{2^{n-1}} \cdot n \\ &= p\left(1 + \frac{2}{2} + \frac{3}{2^2} + \cdots + \frac{n}{2^{n-1}}\right) \approx \frac{1}{2} \cdot 4 = 2 \end{aligned}$$

1.9 n 是奇数时使用二分搜索,时间复杂度为 $O(\log n)$;n 是偶数时使用顺序搜索,时间复杂度为 $O(n)$. 因此以比较作基本运算有 $W(n)=O(n)$. 不能写 $W(n)=\Theta(n)$,因为 n 为奇数时,$\log n$ 不能写成 $\Omega(n)$.

1.10 (1) 以行 3 的除法作为基本运算,行 2 的循环最多执行 $s-1$ 次,于是

$$W(n) = O(s) = O(\sqrt{n})$$

(2) 不能使用 Θ 符号表示算法最坏情况下的时间复杂度,因为对于大素数 n,算法确实需要做$\sqrt{n}$次除法. 但是如果 $n=3k$,k 为任意大的奇整数,那么算法只需 2 次除法;因此不存在常数 c 和 n_0,使得当 $n \geqslant n_0$ 时都有 $c\sqrt{n} \leqslant W(n)$ 成立,即$\sqrt{n}$不是 $W(n)$的渐近的下界.

1.11 **证** 根据 $f=O(h)$和 $g=O(h)$,存在 $c_1>0$ 和正整数 n_1 使得当 $n \geqslant n_1$ 时,有 $f(n) \leqslant c_1 h(n)$. 同理,存在 $c_2>0$ 和正整数 n_2,使得当 $n \geqslant n_2$ 时,有 $f(n) \leqslant c_2 h(n)$. 取 $c=\max\{2c_1, 2c_2\}$,$n_0=\max\{n_1, n_2\}$,那么当 $n \geqslant n_0$ 时,有

$$f(n) + g(n) \leqslant c_1 h(n) + c_2 h(n) \leqslant ch(n)$$

其中 c 为常数,因此 $f+g=O(h)$.

1.12 **证** 因为

$$\lim_{n\to+\infty}\frac{\log_b n}{n^\alpha} = \lim_{n\to+\infty}\frac{\frac{1}{n\ln b}}{\alpha n^{\alpha-1}} = \lim_{n\to+\infty}\frac{1}{\alpha\ln b}\frac{1}{n^\alpha} = 0$$

根据定理 1.1 命题得证.

1.13 **证** 若 $d \leqslant 1$,显然有

$$\lim_{n\to+\infty}\frac{n^d}{r^n} = \lim_{n\to+\infty}\frac{dn^{d-1}}{r^n\ln r} = \frac{d}{\ln r}\lim_{n\to+\infty}\frac{n^{d-1}}{r^n} = 0$$

若 $d>1$,根据洛必达法则,上式中 n^d 将通过逐次求导降低为 $n^{d-1}, n^{d-2}, \cdots$,直到 n^{d_0},其中

$d_0<1$. 而分母的指数函数不变. 根据前面的结果,上式极限为 0. 由定理 1.1 命题得证.

1.14　证　(1) 如果 x 是整数 n,根据定义 $x=x=n$,从而有

$$x-1<x=x=x<x+1$$

如果 $n<x<n+1$,n 为整数,那么 $x=n$,$x=n+1$,从而有

$$x-1<n=x \quad n<x<n+1=x$$
$$\Rightarrow x-1<n=x<x<x=n+1<x+1$$

综合上述命题得证.

(2) 易见,当 x 为整数时命题显然正确,假设 $m<x<m+1$,其中 m 是整数,那么

$$m+n<x+n<m+1+n$$

根据定义有

$$x+n=m+n=x+n$$
$$x+n=m+1+n=x+n$$

(3) 若 $n=2k$,k 为整数,那么

$$\left\lceil\frac{n}{2}\right\rceil+\left\lfloor\frac{n}{2}\right\rfloor=k+k=2k=n$$

若 $n=2k+1$,k 为整数,那么

$$\left\lceil\frac{n}{2}\right\rceil+\left\lfloor\frac{n}{2}\right\rfloor=(k+1)+k=2k+1=n$$

(4) 易见,当 $n=pab$ 时(其中 p 是整数),命题成立. 假设 $n=pab+r$,其中 $0<r<ab$. 那么

$$\left\lceil\frac{\left\lceil\frac{n}{a}\right\rceil}{b}\right\rceil=\left\lceil\frac{\left\lceil\frac{pab+r}{a}\right\rceil}{b}\right\rceil=\left\lceil\frac{pb+\left\lceil\frac{r}{a}\right\rceil}{b}\right\rceil=\left\lceil p+\frac{\left\lceil\frac{r}{a}\right\rceil}{b}\right\rceil=p+\left\lceil\frac{\left\lceil\frac{r}{a}\right\rceil}{b}\right\rceil=p+1$$

上式的最后一步是由于 $0<r<ab$,即 $0<r/a\leqslant b$. 同理也可以得到

$$\left\lceil\frac{n}{ab}\right\rceil=\left\lceil\frac{pab+r}{ab}\right\rceil=\left\lceil p+\frac{r}{ab}\right\rceil=p+\left\lceil\frac{r}{ab}\right\rceil=p+1$$

类似地可以证明命题的另一半.

1.15　(1) $g(n)=O(f(n))$

(2) $f(n)=O(g(n))$

(3) $f(n)=O(g(n))$

(4) $g(n)=O(f(n))$

(5) $f(n)=O(g(n))$

1.16

函数	$f(n)$	$g(n)$	$f(n)=O(g(n))$	$f(n)=\Omega(g(n))$	$f(n)=\Theta(g(n))$
1	$2n^3+3n$	$100n^2+2n+100$	false	true	false
2	$50n+\log n$	$10n+\log\log n$	true	true	true
3	$50n\log n$	$10n\log\log n$	false	true	false
4	$\log n$	$\log^2 n$	true	false	false
5	$n!$	5^n	false	true	false

1.17

$5\log(n+100)^{10}=\Theta(\log n)$， $1+\frac{1}{2}+\cdots+\frac{1}{n}=\Theta(\log n)$， $(\ln n)^2=\Theta(\log^2 n)$，

$\sqrt[3]{n}+\log n=\Theta(\sqrt[3]{n})$， $\log(n^{n+1})=\Theta(n\log n)$， $\log(n!)=\Theta(n\log n)$，

$0.001n^4+3n^3+1=\Theta(n^4)$， $3^n=\Theta(3^n)$， $2^{2n}=\Theta(4^n)$， $(n-2)!=\Theta((n-2)!)$

1.18

$n!$， 2^{2n}， $n2^n$， $(\log n)^{\log n}=n^{\log\log n}$， n^3， $n\log n=\Theta(\log(n!))$，

$n=\Theta(\log 10^n)$， $2^{\log\sqrt{n}}$， $2^{\sqrt{2\log n}}$， $\log n=\Theta\left(\sum_{k=1}^{n}\frac{1}{k}\right)$， $\log\log n$

1.19 (1) 迭代法.

$T(n)=n^2+(n-1)^2+\cdots+2^2+T(1)=1+2^2+\cdots+n^2=\frac{n(n+1)(2n+1)}{6}$

(2) 使用主定理,$a=9,b=3,f(n)=n$,因此

$$T(n)=\Theta(n^{\log_3 9})=\Theta(n^2)$$

(3) 使用递归树,

$$T(n)=cn+\frac{3cn}{4}+\left(\frac{3}{4}\right)^2 cn+\cdots=\left[1+\frac{3}{4}+\left(\frac{3}{4}\right)^2+\cdots\right]cn=\Theta(n)$$

(4) 迭代归纳.

$$\begin{aligned}T(n)&=n\log 3+(n-1)\log 3+\cdots+2\log 3+T(1)\\&=\log 3[n+(n-1)+\cdots+2]+1=\Theta(n^2)\end{aligned}$$

(5) 使用主定理,

$$a=5,\quad b=2,\quad f(n)=n^2\log^2 n=O(n^{\log_2 5-\epsilon}),\quad T(n)=\Theta(n^{\log_2 5})$$

(6) 使用主定理.$a=2,b=2,f(n)=n^2\log n$,取$c=3/4$,则

$$af\left(\frac{n}{b}\right)=2\left(\frac{n}{2}\right)^2\log\left(\frac{n}{2}\right)=\frac{n^2}{2}(\log n-1)\leqslant\frac{n^2}{2}\log n\leqslant cn^2\log n=cf(n)$$

于是 $T(n)=\Theta(n^2\log n)$.

(7) 迭代法.

$$\begin{aligned}T(n)&=T(n-1)+\frac{1}{n}=\cdots=\frac{1}{n}+\frac{1}{n-1}+\cdots+\frac{1}{2}+T(1)\\&=1+\frac{1}{2}+\cdots+\frac{1}{n}=\Theta(\log n)\end{aligned}$$

(8) 迭代法.

$$\begin{aligned}T(n)&=\log n+\log(n-1)+\cdots+\log 2+T(1)\\&=\log(n!)+T(1)=\Theta(n\log n)\end{aligned}$$

1.20 根据主定理有

$$T(n)=\Theta(n^{\log_2 7}),\quad W(n)=\Theta(n^{\log_4 a})$$

若使 $\log_4 a<\log_2 7$,必有

$$\frac{\log_2 a}{\log_2 4}<\log_2 7\Rightarrow\log_2 a<\log_2 7^2=\log_2 49$$

即 $a<49$,于是最大的 $a=48$.

1.21 应该选算法 A. 3 个算法最坏情况下时间复杂度的递推方程及解分别为：
算法 A：$T_A(n)=5T_A(n/2)+O(n)$，　$T_A(n)=\Theta(n^{\log 5})$
算法 B：$T_B(n)=2T_B(n-1)+O(1)$，　$T_B(n)=\Theta(2^n)$
算法 C：$T_C(n)=9T_C(n/3)+O(n^3)$，　$T_C(n)=\Theta(n^3)$
时间复杂度最低的是算法 A.

第 2 章 分治策略

2.1 内 容 提 要

1. 基本概念

分治策略(divided and conquer)是一种算法设计技术,其主要思想是:将原问题划分(或归约)为彼此独立的、规模较小而结构相同的子问题,递归地求解所有的子问题并将子问题的解组合从而得到原问题的解.

分治算法的设计步骤

(1) 分解:将原问题划分或归约为若干个子问题,子问题必须与原问题具有相同的结构,每个子问题可独立求解. 对输入划分时注意保持子问题规模的均衡.

(2) 递归求解:按次序递归求解每个子问题,注意给出递归求解的终止条件,即当子问题规模足够小时直接求解的方法.

(3) 组合:把上述子问题的解组合,从而得到原问题的解.

分治算法的分析方法

列出关于时间复杂度函数(最坏或平均情况下)的递推方程和初值,求解递推方程.

提高分治算法效率的途径

利用子问题之间的依赖关系减少子问题数目.

利用预处理过程减少递归内部的运算量.

2. 与分治算法有关的递推方程及求解

(1) 原问题经归约后,子问题规模减少某个常数,时间复杂度的递推方程形式如下:

$$T(n) = \sum_{i=1}^{k} a_i T(n-i) + f(n)$$

其中 $f(n)$ 为分解子问题和组合子问题的解所做的工作量. 该方程可采用迭代法求解.

(2) 原问题经均衡划分后,产生 a 个子问题,子问题规模为原问题的 $1/b$,时间复杂度函数的递推方程形式如下:

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

其中 $f(n)$ 为分解子问题和组合子问题的解所做的工作量. 该方程可用主定理或递归树求解.

当 $f(n)$ 是常数时,

$$T(n)=\begin{cases}\Theta(n^{\log_b a}) & a\neq 1\\ \Theta(\log n) & a=1\end{cases}$$

当 $f(n)$ 为 cn(c 为常数)时,

$$T(n)=\begin{cases}\Theta(n) & a<b\\ \Theta(n\log n) & a=b\\ \Theta(n^{\log_b a}) & a>b\end{cases}$$

3. 典型的分治算法

二分检索　BinarySearch(T,l,r,x),在数组 $T[l..r]$中查找 x,基本运算为 x 与 T 中元素的比较,最坏情况下的时间为 $W(n)=O(\log n)$.

二分归并排序　MergeSort(A,p,r),对数组 $A[p..r]$的元素进行排序,基本运算为元素比较,最坏情况下的时间为 $W(n)=O(n\log n)$.

芯片测试　Test(n),通过互相测试在 n 片芯片(其中好芯片至少比坏芯片多 1 片)中找出 1 片好芯片,基本运算为测试,最坏情况下的时间为 $W(n)=O(n)$.

大整数相乘　两个 n 位整数相乘,基本运算为位乘,$W(n)=O(n^{\log 3})=O(n^{1.59})$.

Strassen 矩阵乘法　两个 n 阶矩阵相乘,基本运算为矩阵元素相乘,最坏情况下的时间为 $W(n)=O(n^{\log 7})=O(n^{2.8075})$.

最邻近点对　给定平面上的 n 个点,求其中距离最近的 2 个点,基本运算为计算两点间最小距离和比较运算,加预处理后最坏情况下的时间是 $W(n)=O(n\log n)$.

快速排序　Quicksort(A,p,r) 对数组 $A[p..r]$的元素进行排序,基本运算为元素比较,最坏情况下的时间是 $W(n)=O(n^2)$,平均时间是 $A(n)=O(n\log n)$.

选第 k 小　Select(S,k) 在数组 S 中选第 k 小元素,基本运算为元素比较,最坏情况下的时间是 $W(n)=O(n)$.

2.2　习　　题

对于本章与算法设计有关的习题,解题要求如下:先用一段简短的文字说明算法的主要设计思想,其中所引入的符号要给出必要的说明,是否给出伪码根据题目要求确定. 可以调用书上的算法作为子过程,最后对所设计的算法需要给出最坏情况下时间复杂度的分析.

2.1　设输入是 n 个数的数组 $A[1..n]$,下述排序算法是插入排序.

算法 InsertSort(A)

```
1. for i←2 to n do
2.     x←A[i]
3.     j←i-1
4.     while j>0 and A[j]>x do
5.         A[j+1]←A[j]
6.         j←j-1
7.     A[j+1]←x
```

改进上述算法,在插入元素 $A[i]$时用二分查找代替顺序查找,将这个算法记作 ModInsertSort,给出该算法的伪码,并估计算法在最坏情况下的时间复杂度.

2.2 设 A 是 n 个非 0 实数构成的数组,设计一个算法重新排列数组中的数,使得负数都排在正数前面. 要求算法使用 $O(n)$ 的时间和 $O(1)$ 的空间.

2.3 双 Hanoi 塔问题是 Hanoi 塔问题的一种推广,与 Hanoi 塔的不同点在于: $2n$ 个圆盘,分成大小不同的 n 对,每对圆盘完全相同. 初始,这些圆盘按照从大到小的次序从下到上放在 A 柱上,最终要把它们全部移到 C 柱,移动的规则与 Hanoi 塔相同.

(1) 设计一个移动的算法并给出伪码描述.

(2) 计算你的算法所需要的移动次数.

2.4 给定含有 n 个不同的数的数组 $L=\langle x_1,x_2,\cdots,x_n\rangle$. 如果 L 中存在 x_i,使得 $x_1<x_2<\cdots<x_{i-1}<x_i>x_{i+1}>\cdots>x_n$,则称 L 是单峰的,并称 x_i 是 L 的"峰顶". 假设 L 是单峰的,设计一个算法找到 L 的峰顶.

2.5 设 A 是 n 个不同的数排好序的数组,给定数 L 和 U,$L<U$,设计一个算法找到 A 中满足 $L<x<U$ 的所有的数 x.

2.6 设 $\boldsymbol{M}$ 是一个 n 行 n 列的 0-1 矩阵,每行的 1 都排在 0 的前面.

(1) 设计一个最坏情况下 $O(n\log n)$ 时间的算法找到 $\boldsymbol{M}$ 中含有 1 最多的行,说明算法的设计思想,估计最坏情况下的时间复杂度.

(2) 对上述问题,能否找到一个最坏情况下 $O(n)$ 时间的算法?

2.7 设 A 是含有 n 个元素的数组,如果元素 x 在 A 中出现的次数大于 $n/2$,则称 x 是 A 的主元素.

(1) 如果 A 中元素是可以排序的,设计一个 $O(n\log n)$ 时间的算法,判断 A 中是否存在主元素.

(2) 对于(1)中可排序的数组,能否设计一个 $O(n)$ 时间的算法?

(3) 如果 A 中元素只能进行"是否相等"的测试,但是不能排序,设计一个算法判断 A 中是否存在主元素.

2.8 设 A 和 B 都是从小到大已经排好序的 n 个不等的整数构成的数组,如果把 A 与 B 合并后的数组记作 C,设计一个算法找出 C 的中位数.

2.9 在 $n(n\geqslant 3)$ 枚硬币中有一枚重量不合格的硬币(重量过轻或者过重),如果只有一架天平可以用来称重且称重的硬币数没有限制,设计一个算法找出这枚不合格的硬币,使得称重的次数最少? 给出算法的伪码描述. 如果每称 1 次就作为 1 次基本运算,分析算法的最坏情况下的时间复杂度.

2.10 考虑下述 n 阶矩阵乘法的分治算法: 将 $n\times n$ 矩阵顺序划分成 $n/3\times n/3$ 块,每块为 3×3 的小矩阵。假设两个 3×3 的小矩阵相乘可以用 k 次数的乘法运算完成(k 为固定正整数),原来规模为 n 的问题就可以归结为规模为 $n/3$ 的子问题. 设上述算法的时间复杂性函数为 $T(n)$.

(1) 列出关于 $T(n)$ 的递推方程并估计 $T(n)$ 的阶.

(2) k 最大取什么值能够使得 $T(n)=o(n^{\log 7})$?

2.11 设 $P(x)=a_0+a_1x+a_2x^2+\cdots+a_{n-1}x^{n-1}+x^n$ 是最高次项系数为 1 的 n 次多项式,使得 $P(x)=0$ 的数 x 称为该多项式的根. 假设存在算法 A 和 B,其中 A 可以在 $O(k)$ 时间内计算一个 k 次多项式与一个 1 次多项式的乘积;B 可以在 $O(i\log i)$ 时间内计算两个 i 次多项式的乘积. 利用算法 A 和 B 设计一个分治算法确定以给定整数 $d_1,d_2,\cdots,d_n$ 为根的

n 次多项式 $P(x)$.

2.12 设 $A=\{a_1,a_2,\cdots,a_n\}$，$B=\{b_1,b_2,\cdots,b_m\}$是整数集合，其中 $m=O(\log n)$. 设计一个算法找出集合 $C=A\cap B$. 要求给出伪码描述.

2.13 考虑算法 2.11 的 Select 算法.

(1) 如果初始的元素分组采用 3 个一组，算法的时间复杂度将是多少？

(2) 如果初始的元素分组采用 7 个一组，算法的时间复杂度将是多少？

2.14 设 S 是 n 个不等的正整数的集合，n 为偶数. 给出一个算法将 S 划分成子集 S_1 和 S_2，使得 $|S_1|=|S_2|=n/2$，且

$$\left|\sum_{x\in S_1}x-\sum_{x\in S_2}x\right|$$

达到最大，即使得两个子集元素之和的差达到最大.

2.15 给定 n 个不同数的数组 S 和正整数 i，$i\leqslant n^{1/2}$，求 S 中最大的 i 个数，并且按照从大到小的次序输出. 有下述算法：

算法 A：调用 i 次找最大算法 Findmax，每次从 S 中删除一个最大的数.

算法 B：对 S 排序，并输出 S 中最大的 i 个数.

(1) 分析 A、B 两个算法在最坏情况下的时间复杂度.

(2) 试设计一个最坏情况下时间复杂度的阶更低的算法，要求给出伪码.

2.16 设 S 为 n 个不同数的集合，

(1) 设计算法找出 S 中的数 x 和 y 使得 $\forall u,v\in S$，$|x-y|\geqslant|u-v|$.

(2) 设计算法找出 S 中的数 x 和 y 使得 $\forall u,v\in S$，$|x-y|\leqslant|u-v|$.

2.17 给定 n 个不等的整数构成的集合 L 和整数 s，设计一个算法判断在 L 中是否存在两个整数 x 和 $y(x<y)$，满足 $x+y=s$. 以加法运算作为基本运算分析你的算法在最坏情况下的时间复杂度.

2.18 设平面直角坐标系中有 n 个点 (x_1,y_1)，(x_2,y_2)，$\cdots$，(x_n,y_n)，每个点到原点 $(0,0)$ 的距离彼此不等. 设计一个算法找到距离原点 $(0,0)$ 最近的 $\sqrt{n}$ 个点，并按照距原点从远到近的顺序输出点的标号. 要求给出伪码描述.

2.19 设 A 是 n 个不等的整数数组，$n=2^k$，设计一个分治算法找出 A 中的最大数 max 和最小数 min. 要求给出伪码描述.

2.20 有 n 个人，其中某些人是诚实的，其他人可能会说谎. 现在需要进行一项调查，该调查由一系列测试构成. 每次测试如下进行：选 2 个人，然后提问：对方是否诚实？每个人的回答只能是"是"或者"否". 假定在这些人中，所有诚实的人回答都是正确的，而其他人的回答则不能肯定是否正确. 如果诚实的人数 $>n/2$，试设计一个调查算法，以最小的测试次数从其中找出一个诚实的人.

2.21 多选问题：设 S 是 n 个不等的数的集合，对于给定的 $1\leqslant r\leqslant n$，$K=\{k_1,k_2,\cdots,k_r\}$是 r 个正整数的集合，其中 $1\leqslant k_1<k_2<\cdots<k_r\leqslant n$. 请设计算法在 S 中找出第 k_1 小、第 k_2 小、…、第 k_r 小的 r 个元素. 例如 $r=3$，$K=\{2,4,7\}$，则输出 S 的第 2 小、第 4 小、第 7 小的元素. 不难看出，当 $r=1$ 时，该问题就是一般的选择问题，当 $r=n$ 时，该问题就变成了排序问题.

(1) 设计一个最坏情况下时间复杂度为 $O(nr)$ 的算法.

(2) 若 $r>1$，设计一个时间复杂度为 $O(n\log r)$ 的算法.

2.22 设 $A=\{a_1,a_2,\cdots,a_n\}$, $B=\{b_1,b_2,\cdots,b_m\}$是整数集合,其中 $m=O(\log n)$. 设计算法计算集合 $C=(AB)\cup(BA)$,说明算法的主要步骤,并以比较作基本运算分析算法最坏情况下的时间复杂度.

2.23 设 A 是 n 个数的序列,如果 A 中的元素 x 满足以下条件:小于 x 的数的个数$\geqslant n/4$,且大于 x 的数的个数$\geqslant n/4$,则称 x 为 A 的近似中值. 设计算法求出 A 的一个近似中值. 说明算法的设计思想和最坏情况下的时间复杂度.

2.24 设 A 是 n 个数构成的数组,其中出现次数最多的数称为众数,设计一个算法求 A 的众数,给出伪码和最坏情况下的时间复杂度.

2.25 一个实验可得出 n 个不同的值,分别用数 x_1, x_2, $\cdots$, x_n表示. 已知 x_i出现的概率是 p_i,$i=1,2,\cdots,n$,且所有概率之和等于1. 试设计一个算法找到值 x_k,使得所有小于 x_k的值出现的概率之和不超过1/2,且所有大于 x_k的值出现的概率之和也不超过1/2. 例如,实验结果为:$x_1=2$, $x_2=1$, $x_3=4$, $x_4=3$, $x_5=5$,出现的概率依次为:$p_1=0.2$, $p_2=0.4$, $p_3=0.1$, $p_4=0.1$, $p_5=0.2$,那么 x_1就是所求的值. 比 x_1小的数只有 x_2,它的概率是0.4;比 x_1大的数有 x_3, x_4和 x_5,它们的概率之和是0.4. 说明算法的设计思想,估计算法最坏情况下的时间复杂度.

2.26 某石油公司计划建造一条由东向西的输油管道,该管道要穿过一个有 n 口油井的油田. 从每口油井都要有一条输油管道沿最短路径(南北方向)与主管道相连. 如果给定 n 口油井的位置,即它们的 x 坐标和 y 坐标. 设计一个算法来确定主管道的位置,使得每口油井到主管道之间的输油管道长度之和达到最小?

2.27 如图2.1(注:主教材图2.8)所示,城市街道都是水平或垂直分布,有 $m+1$ 条,不妨设任何两个相邻位置之间的距离都是1. 在街道的十字路口有 n 个商店,图中的 $n=3$,$m=8$,3个商店的坐标位置(图中的圆点)分别是(2,4),(5,3),(6,6). 现在需要在某个路口位置建一个合用的仓库. 若仓库选择(3,5)位置,那么这3个商店到仓库的路程(只能沿街道行进)总长至少是10. 设计一个算法找到仓库的最佳位置,使得所有商店到仓库路程的总长达到最短.

图2.1 街道图

2.28 在Internet上的搜索引擎经常需要对信息进行比较,比如可以通过某个人对一些事物的排名来估计他(或她)对各种不同信息的兴趣,从而实现个性化的服务. 对于不同的排名结果可以用逆序来评价它们之间的差异. 考虑 $1,2,\cdots,n$ 的排列 $i_1 i_2\cdots i_n$,如果其中存在 i_j,i_k,使得 $j<k$ 但是 $i_j>i_k$,那么就称(i_j,i_k)是这个排列的一个逆序. 一个排列含有逆序的个数称为这个排列的逆序数. 例如排列2 6 3 4 5 1含有8个逆序(2,1),(6,3),(6,4),(6,5),(6,1),(3,1),(4,1),(5,1),它的逆序数就是8. 显然,由 $1,2,\cdots,n$ 构成的所有 $n!$个排列中,最小的逆序数是0,对应的排列就是 $12\cdots n$;最大的逆序数是 $n(n-1)/2$,对应的排列就是 $n(n-1)\cdots 21$. 逆序数越大的排列与原始排列的差异度就越大.

利用二分归并排序算法设计一个计数给定排列逆序的分治算法,并对算法进行时间复杂度的分析.

2.29 每个螺母需要和1个螺栓配套使用. 现在有 n 种不同尺寸的螺母与螺栓,恰好可以配成 n 套. 设计一个平均复杂度最低的算法,尽快为每个螺母找到与之配套的螺栓. 该算法的1次基本运算是:把1个螺母尝试与1个螺栓匹配,看看它们是否合适. 尝试结果只能是以下三种之一:螺母大于螺栓,螺母小于螺栓,或者恰好配套. 注意,该算法不能比较2个螺栓的大小,也不能比较2个螺母的大小.

(1) 用文字说明算法的设计思想;

(2) 分别给出算法在最坏和平均情况下的时间复杂度 $W(n)$ 和 $A(n)$.

2.30 对玻璃瓶做强度测试,设地面高度为0,从0向上有 n 个高度,记为 $1,2,\cdots,n$,其中任何两个高度之间的距离都相等. 如果一个玻璃瓶从高度 i 落到地上没有摔破,但从高度 $i+1$ 落到地上摔破了,那么就将玻璃瓶的强度记为 i.

(1) 假设每种玻璃瓶只有1个测试样品,设计算法来测出每种玻璃瓶的强度. 以测试次数作为算法最坏情况下的时间复杂度量度,对算法最坏时间复杂度作出估计.

(2) 假设每种玻璃瓶有足够多的相同的测试样品,设计算法使用最少的测试次数来完成测试. 该算法最坏情况下的时间复杂度是多少?

(3) 假设每种玻璃瓶只有2个相同的测试样品($k=2$),设计次数尽可能少的算法完成测试. 你的算法最坏情况下的时间复杂度是多少?

(4) 尝试将(3)中的方法推广到其他任意给定的 $k(k>2)$ 的情况.

2.3 习题解答与分析

2.1 算法的伪码是:

```
ModInsertSort(A[1..n])
for i←2 to n do
    x←A[i]
    k←BinarySearch(A[1..i−1],x)
                                   //在A[1..i−1]中二分查找x应该插入的位置k
    for j←i−1 downto k
        A[j+1]←A[j]
    A[k]←x
```

for 循环执行 $n-1$ 次,在 $i-1$ 规模的数组二分查找的比较次数为 $\log(i-1)+1$,因此总的比较次数为

$$\begin{aligned}\sum_{i=2}^{n}(\log(i-1)+1) &= n-1+\sum_{i=1}^{n-1}\log i \leqslant n-1+\sum_{i=1}^{n-1}\log i \\ &\leqslant n-1+\int_{1}^{n}\log x\,\mathrm{d}x \\ &= n-1+n\log n-n\log \mathrm{e}+\log \mathrm{e} \\ &= O(n\log n)\end{aligned}$$

2.2 类似快速排序的划分过程. 从后向前把每个数与0比较,找到第一个负数 $A[p]$. 从前向后把每个数与0比较,找到第一个正数 $A[q]$,如果 $p>q$,则将 $A[p]$ 与 $A[q]$ 交换. 交

换后如果 $p-q=1$,算法停止;否则继续这个过程,或者找到下一对可以交换的数,或者扫描到 $p\leqslant q$ 停止.

2.3 (1) 算法设计思想:分治策略.先递归地将上面的 $2(n-1)$ 个盘子从 A 柱移到 B 柱;用 2 次移动将最大的 2 个盘子从 A 柱移到 C 柱;递归地将 B 柱的 $2(n-1)$ 个盘子从 B 柱移到 C 柱.

伪码描述是:

```
BiHanoi(A,C,n)                    //从 A 到 C 移动 2n 只盘子
1. if n=1 then BiMove(A,C)        //从 A 到 C 移动 2 只盘子
2. else BiHanoi(A,B,n-1)
3.      BiMove(A,C)
4.      BiHanoi(B,C,n-1)
```

(2) 设 $2n$ 个圆盘的移动次数是 $T(n)$,则第 2 行和第 4 行的递归调用的子问题规模是 $n-1$,第 3 行是 2 次移动,于是有

$$\begin{cases} T(n) = 2T(n-1)+2 \\ T(1) = 2 \end{cases}$$

解得 $T(n)=2^{n+1}-2$.

2.4 因为 L 中存在峰顶元素,因此 $|L|\geqslant 3$.使用二分查找算法.如元素数等于 3,则 $L[2]$是峰顶元素.当元素数 n 大于 3 时,令 $k=n/2$,比较 $L[k]$与它左边和右边相邻的项.如果 $L[k]>L[k-1]$且 $L[k]>L[k+1]$,则 $L[k]$为峰顶元素;否则,如果 $L[k-1]>L[k]>L[k+1]$,则继续搜索 $L[1..k-1]$的范围;如果 $L[k-1]<L[k]<L[k+1]$,则继续搜索 $L[k+1..n]$的范围.每比较 2 次,搜索范围减半,直到元素数小于等于 3 停止递归调用.时间复杂度函数为

$$\begin{cases} T(n) = T(n/2)+2 \\ T(1) = c,\ c \text{ 为某个常数} \end{cases}$$

根据主定理,$T(n)=O(\log n)$.

2.5 算法设计思想是:在 A 中使用二分查找算法找 L.如果 $L=A[i]$,找到 L 的位置 i,然后把 i 加 1;如果 L 不在 A 中,那么找到大于 L 的最小数的位置 i.类似地,在 A 中二分查找 U,找到 U 所在的位置 j,然后把 j 减 1;如果 U 不在 A 中,那么找到小于 U 的最大数的位置 j.输出 A 中从 i 到 j 的全体数.

二分查找 L 和 U 的时间复杂度都是 $O(\log n)$,于是算法的时间复杂度是 $O(\log n)$.

2.6 (1) 分治策略,在归约为子问题时可以有几种不同的方法.

方法一:顺序对每一行进行二分检索,找到最后一个 1 的位置,用 y 和 max 记录至今为止含有 1 最多的行号和该行的 1 的个数.二分检索时间为 $O(\log n)$,n 行的总时间为$O(n\log n)$.

方法二:对列号进行二分检索,找到最大的含有 1 的列号.

初始令 $i=1,j=n$,检索范围为列 i 到列 j.取 $k=(i+j)/2$,从上到下顺序检查第 k 列是否含有 1.如果含有 1,下面的搜索范围将缩小到第 k 列到第 j 列;否则,搜索范围缩小到第 i 列到第 $k-1$ 列.不管如何,搜索范围至少减半.由于每次在第 k 列的比较次数至多是 $O(n)$,最坏情况下的时间复杂度有下述递推方程:

$$W(t) = W(t/2)+O(n)$$

$$W(1) = O(n)$$

其中 t 表示子问题规模. 初始时 $t=n$. 不妨令 $n=2^k$，根据 k 次迭代得到 $t=1$，每次迭代的工作量是 $O(n)$，因此

$$W(n) = kO(n) = O(n\log n)$$

方法三：将整个区域划分成上下相等的两部分，即：第 1 行到第 $n/2$ 行，第 $n/2+1$ 行到第 n 行，从而构成两个规模近似相等的子问题. 分别找出每个子问题的含 1 最多的行，比如说行 i 和行 j，记录它们含有 1 的个数. 然后把行 i 和行 j 含有 1 的个数进行比较，找到其中含 1 较多的行即可. 当递归执行到规模为 1 的子问题时，由于只有一行，可以采用二分检索的方法确定其含有 1 的个数.

最坏情况下的时间复杂度满足下述递推方程：

$$W(n) = 2W(n/2) + 1$$
$$W(1) = \log n$$

不妨设 $n=2^k$，根据迭代有

$$\begin{aligned} W(n) &= 2W(n/2) + 1 = 2^2W(n/4) + 2 + 1 \\ &= 2^3W(n/2^3) + 2^2 + 2 + 1 \\ &\vdots \\ &= 2^kW(n/2^k) + 2^{k-1} + \cdots + 2 + 1 \\ &= 2^kW(1) + 2^k - 1 = n\log n + n - 1 = O(n\log n) \end{aligned}$$

(2) 设含有 1 最多的行中最右边的 1 位于第 y 列(注意第 y 列可能不止一个 1). 下面设计一个 $O(n)$时间的算法，直接找到第 y 列上的某一个 1，并输出这个 1 所在的行号. 这就是从左上角到右下方沿折线搜索的方法. 把矩阵 M 看成由 $n\times n$ 个位置构成的区域，每个位置含有 1 或者 0. $M[i,j]$表示第 i 行第 j 列的元素. 算法从 $M[0,0]$开始，沿第一行向右扫描，直到遇到第一个 0 为止. 设这个 0 在第 j 列. 从这个 0 改变搜索方向，沿第 j 列向下找第一个 1. 设这个 1 在第 k 行. 在这个位置算法改变搜索方向，沿着第 k 行向右搜索，以此类推，直到走到最下面一行或者到达最右边一列的 1 停止. 扫描过程中，每当从向右转到向下搜索时，记下转变前的那一行最右边 1 的位置和行号；每当从向下转到向右搜索时，记下转变后的行的行号.

图 2.2 给出了一个具体的实例，输出 $i=5$. 图中的箭头表示搜索的方向.

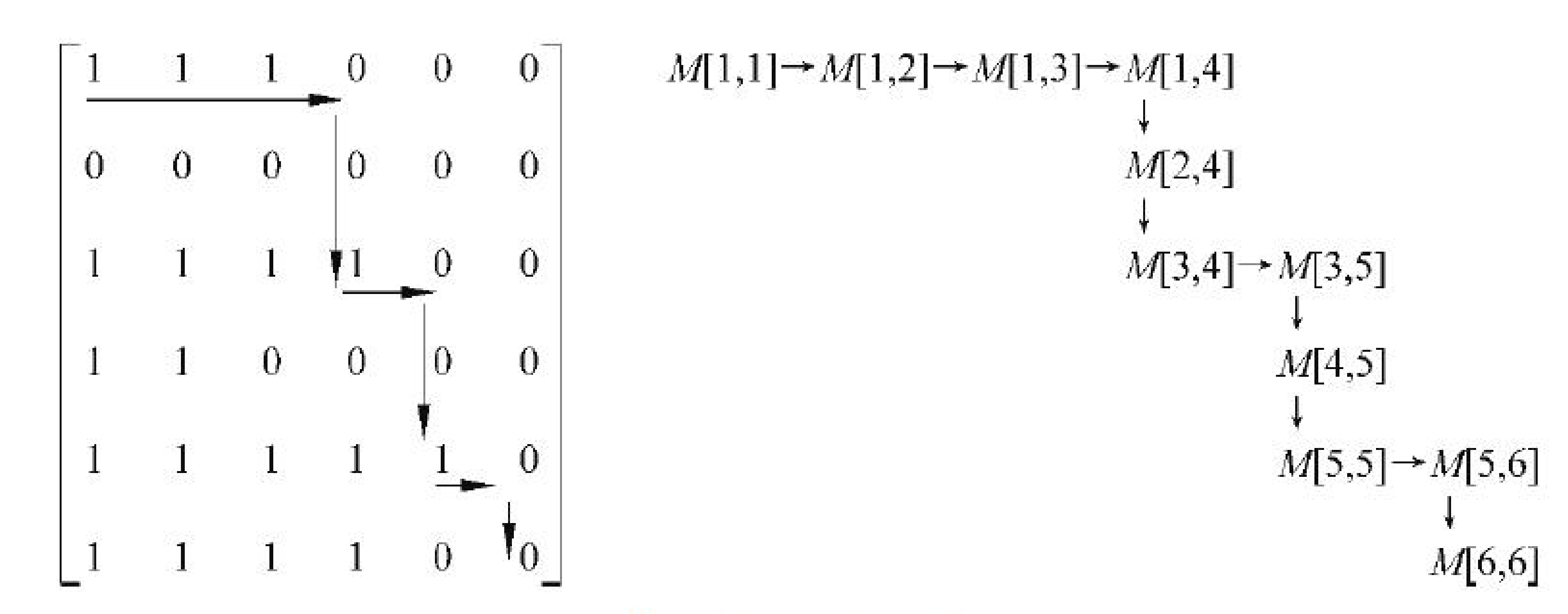


图 2.2　运行实例

从初始位置 $M[1,1]$到某个终止位置 $M[l,n]$或 $M[n,k]$，这条路径所经历的位置至多 $2n-1$ 个. 在每个位置，算法做 1 次比较. 因为算法所做的比较次数不超过 $2n-1$ 次，这就

证明了算法在最坏情况下的时间复杂度是 $O(n)$.

2.7 (1) **方法一**:基于排序的算法.

设计思想:把 A 按照从小到大的顺序排序. 那么相同的元素是连续分布的. 令 x 为 A 中最多的元素. 初始 $x=A[1]$,顺序检查 A 中元素,计数 x 的个数. 如果发现比 x 更多的新元素,就替换 x. 检索完成后,若 x 的个数大于 $n/2$,x 就是主元素;否则 A 中没有主元素.

该算法的时间包括排序时间和通过比较进行计数的时间,总计为

$$T(n)=O(n\log n)+O(n)=O(n\log n)$$

方法二:分治算法.

命题 2.1 若 A 中存在主元素,则将 A 从中间划分为两部分,A 的主元素也必为两部分中至少一部分的主元素.

证 设 x 是 A 的主元素,那么将 A 划分为 L_1 和 L_2 后,$|L_1|=n/2$,$|L_2|=n/2$. L_1 中含有 i 个 x,L_2 中含有 j 个 x,且 $i\leqslant n/2\ /2$,$j\leqslant n/2/2$,那么 x 在 A 中的个数至多为

$$i+j\leqslant\frac{n/2}{2}+\frac{n/2}{2}=\left(\left\lfloor\frac{n}{2}\right\rfloor+\left\lceil\frac{n}{2}\right\rceil\right)\Big/2=n/2$$

与 x 是 A 的主元素矛盾.

分治算法的主要步骤是:

1. 若 A 只含一个元素,则此元素就是主元素,返回此数.

2. 将 A 分为大致相等的两部分 L_1 和 L_2,分别递归调用此算法求其主元素 m_1 和 m_2.

3. 若 m_1 和 m_2 都存在且相等,则这个数就是 A 的主元素,返回此数.

4. 若 m_1 和 m_2 都不存在,则 A 无主元素.

5. 若 m_1 和 m_2 都存在但不等,或者只有一个存在,则分别检查这些数在 A 中出现的次数,如果超过 $n/2$,则是主元素;否则不存在.

该算法的时间分析:第2行的两次递归调用,其中子问题规模是原来规模的一半;第5行的检查时间为 $O(n)$,于是得到时间复杂度的递推方程

$$\begin{cases}T(n)=2T\left(\dfrac{n}{2}\right)+O(n)\\T(1)=0\end{cases}$$

该方程的解是 $T(n)=O(n\log n)$.

(2) $O(n)$时间的算法.

适用于元素可排序的数组.

命题 2.2 A 中的主元素一定是中位数.

证 按照从小到大排序 A 的元素. 若存在主元素 x,那么 x 应该至少占有连续的 $n/2+1$ 个位置. 如果其中不包含中位数的位置在内,那么这些连续位置只能分布在中位数的单侧. 中位数任何一侧的元素数都不超过 $n/2$. 这与主元素的定义矛盾.

算法描述:

1. 找出 A 的中位数 x.

2. 把 x 与 A 中元素顺序比较并计数 x 在 A 中出现的次数 j. 如果 $j>n/2$,则 x 就是主元素;否则不存在.

该算法的时间是第 1 行找中位数的时间和第 2 行的检查时间，两者都是 $O(n)$，于是

$$T(n) = O(n) + O(n) = O(n)$$

(3) 如果 A 中元素不能排序，可采用基于两两元素的比较从而把非主元素淘汰的算法. 其实现方法有两种.

方法一：芯片测试算法. 其设计思想如下：

算法 Delete(A)

1. 如果 $|A| \leqslant 1$ 则结束；否则将 A 的元素两两分组.

2. 每组内两个元素进行比较；如果相等则把 1 个放到 B 中；否则全部淘汰.

3. $A \leftarrow B$ 并转 1.

运行上述 Delete 淘汰过程，如果最后没有元素留下，那么没有主元素. 如果剩下 1 个元素，接着检查该元素在 A 中出现的次数. 若次数大于 $n/2$，则是主元素；否则没有主元素.

下面证明算法的正确性.

命题 2.3 假设 n 是偶数，经过一次 Delete(A)的淘汰过程，A 具有以下性质.

性质 1：如果 A 中含有主元素，淘汰后的主元素数仍旧多于非主元素数.

性质 2：A 的元素至多为原来的一半.

证 因为 n 是偶数，在分组中没有轮空的元素. 考虑分组的 3 种类型.

① 含有两个主元素：有 a 个组.

② 含有一个主元素和一个非主元素：有 b 个组.

③ 含有两个非主元素：有 c 个组.

淘汰前的主元素数为 $2a+b$，非主元素数为 $2c+b$. 因为 $2a+b>2c+b$，从而有 $a>c$. 根据 Delete 第 2 行的淘汰规则，在 1 次淘汰后，主元素数为 a，非主元素数至多为 c，由于 $a>c$，淘汰后主元素数仍旧多于非主元素数.

此外，由于每组至少淘汰 1 个元素，所以剩下的元素数至多是原来的一半.

当 n 是奇数时，如果被轮空的元素不是主元素时，有可能淘汰后的主元素数等于非主元素数. 比如数组元素为 1,1,1,2,3，分组是{1,1}，{1,2}，3 被轮空，经过 1 轮 Delete 淘汰后，剩下的是 1,3. 可以用下述办法解决这个问题：检查轮空元素在 A 中出现的次数，如果该元素是主元素，则算法结束；如果不是，则在下一轮 Delete 过程开始之前先把这个元素淘汰掉，从而保证每次进入 Delete 过程的剩余数组都满足“主元素数大于非主元素数”的性质.

由命题 2.3 可以断定，如果 A 有主元素，它一定会在所有的 Delete 过程结束后留下来，算法的正确性就有了保证. 需要说明的是：“留下来”是主元素的必要条件，但不是充分条件，可能有非主元素也会留下来. 比如输入是 1,2,3,4,5,6,7,7，没有主元素. 如果第一次分组是{1,2}，{3,4}，{5,6}，{7,7}，那么一轮 Delete 淘汰后只剩下 7，但 7 不是主元素. 于是算法必须对淘汰后剩下的元素进行验证，即检查它在原始数组 A 中出现的次数.

最后考虑算法的运行时间，经过 1 次 Delete 过程子问题规模至少减半，Delete 过程需要比较 $n/2$ 次，加上对轮空元素的检查也不超过 $O(n)$次，于是得到下述递推方程：

$$\begin{cases} W(n) = W\left(\dfrac{n}{2}\right) + O(n) \\ W(1) = 0 \end{cases}$$

与芯片测试问题一样,该方程的解是 $W(n)=O(n)$.

方法二:栈淘汰方法.

设计思想:设立一个栈 S,大小为 $n/2$. 下面顺序对 A 中的元素进行处理. 对任何元素 x,处理原则是:

1. 如果栈为空,则 x 进栈.

2. 如果栈不空,则比较 x 与栈顶元素. 如果相等,则 x 进栈;否则 x 与栈顶元素一起淘汰.

在 A 中元素全部处理完成后,如果栈内剩有元素,那么检查栈顶元素在 A 中出现的次数. 次数大于 $n/2$ 的是主元素;否则没有主元素. 如果栈为空,则没有主元素.

为说明算法的正确性,需要下面的命题.

命题 2.4 在上述栈操作后,栈具有下述性质.

性质 1:如果栈不空,那么其中的元素都相等.

性质 2:如果 A 中有主元素,那么它一定留在栈中.

证 假设栈内剩余的元素不等,那么一定存在两个相邻的元素不等,比如 x 和 y,y 在 x 的上面. y 进栈时,x 为当时的栈顶元素,这与栈的操作规则 2 矛盾.

假设 x 为 A 中的主元素,x 的个数一定大于 $n/2$. 如果 x 在进栈时被淘汰,根据规则 2,当时栈顶元素不是主元素,而这个元素也将同时被淘汰. 于是,要淘汰所有的主元素,至少需要多于 $n/2$ 个非主元素,这与 A 中元素总数等于 n 矛盾.

与前面的情况类似,主元素一定留在栈里,但留在栈里的有可能是非主元素. 因此需要对留下的元素检查其在 A 中出现的次数.

该算法的时间是 $W(n)=O(n)+O(n)=O(n)$.

2.8 设计思想:分治算法.

设 $A[0..n-1]$ 和 $B[0..n-1]$ 是输入数组,令 $k=n/2$,取 $A[k]$ 和 $B[k]$ 比较. 如果 $A[k]=B[k]$,那么 $A[k]$ 就是所求的中位数. 如果 $A[k]<B[k]$,那么 $A[0..k-1]$ 和 $B[k+1..n-1]$ 不含 C 的中位数. 下一步递归查找的子问题是 $A[k..n-1]$ 和 $B[0..k]$,子问题规模减半(至多差常数 1). 类似地,如果 $A[k]>B[k]$,那么 $A[k+1..n-1]$ 和 $B[0,k-1]$ 不含有 C 的中位数. 剩下的子问题规模也减半. 于是可以递归对子问题处理. 如果 $n=1$,A 与 B 中各有 1 个数,那么规定两个数中较大的一个就是所求的中位数.

该算法的时间复杂度满足递推方程

$$\begin{cases} W(n) = W\left(\dfrac{n}{2}\right)+1 \\ W(1) = 1 \end{cases}$$

该方程的解是 $W(n)=O(\log n)$.

2.9 算法设计思想:采用分治策略. 如果 $n<3$,那么将剩下的硬币与拿走的硬币(合格硬币)比较,不等的那枚就是不合格的硬币;否则将 n 枚硬币分成大致相等的 3 份,如果 $n(\bmod 3)\neq 0$,那么令两份少的硬币相等. 取两份相等的硬币放到天平上,如果天平不平衡,那么这两份里包含着不合格的硬币;否则不合格的硬币在剩下的一份中. 递归对归约后的子问题进行处理. 直到 $n<3$ 为止.

算法 Coin(A,n) //A 是 n 个硬币的集合

1. $k \leftarrow n/3$
2. 将 A 中硬币分成 X, Y, Z 三个集合，使得 $|X|=|Y|=k, |Z|=n-2k$
3. 　if $W(X) \neq W(Y)$　　　　　　//$W(X), W(Y)$ 分别为 X 或 Y 的重量
4. 　then $A \leftarrow X \cup Y$
5. 　else $A \leftarrow Z$
6. 　$n \leftarrow |A|$
7. if $n>2$ then goto 1
8. else 将 A 中的硬币与拿走的硬币比较．如果不等，则是不合格硬币．

该算法有下述递推方程：

$$\begin{cases} T(n) = T(2n/3) + O(1) \\ T(1) = 0, T(2) = 1 \end{cases}$$

该方程的解是 $T(n)=O(\log n)$．

如果将硬币分成个数大致相等的 4 份，取个数相等的两份放到天平上．如果天平平衡，那么不合格的硬币在剩下的硬币中；否则在被称重的硬币中．1 次称重后待检测的硬币数减半，直到边界条件为止．这种分治算法的时间也是 $O(\log n)$．

2.10　(1) 设 $A=(a_{ij})_n$ 与 $B=(b_{ij})_n$ 是两个输入的 n 阶矩阵，将 A 和 B 都划分成 $n/3 \times n/3$ 块，每块是 3×3 的小矩阵．设划分后的矩阵分别记为 $A'=(A_{sl})_{n/3}$ 与 $B'=(B_{sl})_{n/3}$，其中 A' 与 B' 的元素 A_{sl} 与 B_{sl} 是 3×3 小矩阵，那么 $C=AB=A'B'=(C_{sl})_{n/3}$，其中 C_{sl} 也是 3×3 小矩阵，且

$$C_{sl} = \sum_{t=1}^{n/3} A_{st} B_{tl}$$

于是 AB 的计算可以看做 $A'B'$ 的计算，这是一个输入规模为 $n/3$ 的子问题．设原问题的计算时间是 $T(n)$，其中 1 次基本运算是矩阵元素 a_{ik} 和 b_{kj} 的 1 次乘法．在子问题的计算中每个 C_{sl} 由若干个 A_{st} 和 B_{tl} 的相乘而得到，因此子问题的基本运算是形如 A_{st} 和 B_{tl} 的两个 3×3 矩阵的 1 次相乘．这是不同的基本运算．根据已知条件，两个 3×3 矩阵相乘可以用 k 次原始矩阵的元素乘法来实现．换句话说，子问题的 $T(n/3)$ 个 A_{st} 与 B_{tl} 的相乘相当于 $kT(n/3)$ 个原始矩阵元素相乘．于是得到递推方程

$$\begin{cases} T(n) = kT(n/3) \\ T(3) = k \end{cases}$$

利用主定理，该方程的解是 $T(n)=\Theta(n^{\log_3 k})$．

(2) 要使 $T(n)=o(n^{\log 7})$，就要满足 $\log_3 k < \log 7$，即 $k < 3^{\log 7}$，于是 k 最大的值是 21．

2.11　相当于计算多项式乘积 $(x-d_1)(x-d_2)\cdots(x-d_n)$．用分治法，将多项式划分成大小均衡的两半，每部分分别相乘，然后将所得结果进一步相乘．若 n 为偶数，则将 $P(x)$ 分为两部分：

$$P_1(x) = (x-d_1)(x-d_2)\cdots(x-d_{n/2})$$

$$P_2(x) = (x-d_{n/2+1})(x-d_{n/2+2})\cdots(x-d_n)$$

若 n 为奇数，则将 $P(x)$ 分为三部分：

$$P_3(x) = (x-d_1)(x-d_2)\cdots(x-d_{(n-1)/2})$$

$$P_4(x) = (x-d_{(n+1)/2})(x-d_{(n+1)/2+1})\cdots(x-d_{n-1})$$

$$P_5(x) = (x - d_n)$$

算法的设计思想是：

1. 如果 n 是偶数，分别对 $P_1(x)$ 与 $P_2(x)$ 递归计算，然后计算 $P_1(x)P_2(x)$.

2. 若 n 是奇数，分别对 $P_3(x)$ 与 $P_4(x)$ 递归计算，然后计算 $P_3(x)P_4(x)P_5(x)$.

$P_1(x)$ 与 $P_2(x)$ 是 $n/2$ 次多项式，$P_3(x)$ 与 $P_4(x)$ 是 $(n-1)/2$ 次多项式，这 4 个子问题规模都不超过原问题规模的一半. 计算 $P_1(x)P_2(x)$，根据算法 B 需要 $O\left(\frac{n}{2}\log\frac{n}{2}\right)$ 时间. 计算 $P_3(x)P_4(x)P_5(x)$，先使用算法 B 计算 $P_3(x)P_4(x)$，需要 $O\left(\frac{n}{2}\log\frac{n}{2}\right)$ 时间；然后使用算法 A 乘以 $P_5(x)$，需要 $O(n)$ 时间，于是算法最坏情况下的时间复杂度是

$$T(n) \leqslant 2T\left(\frac{n}{2}\right) + O\left(\frac{n}{2}\log\frac{n}{2}\right) + O(n) = 2T\left(\frac{n}{2}\right) + O(n\log n)$$
$$= O(n\log^2 n)$$

2.12 算法设计思想：在 A 与 B 中选择一个数组排序，然后顺序对另一个数组的每个元素使用二分查找算法，看看它在这个数组中是否出现. 如果出现，则将它放入 C. 关键是选择 A 还是 B 进行排序. 算法的主要时间消耗是排序. 因此选择较小的数组进行排序是合适的. 考虑到 $|B| = m = O(\log n)$，比起 $|A|$ 要小得多，因此对 B 排序.

算法

输入：数组 $A[1..n]$ 和 $B[1..m]$

输出：数组 C，使得 $C = A \cap B$

```
1. C←∅
2. L←Sort(B)                    //对 B 排序
3. for i←1 to n do
4.     x←A[i]
5.     j←BinarySearch(L,x)  //在 L 中二分搜索 x，若 x 在 L 中，j 为序标；否则 j 为 0
6.     if j>0
7.     then C←C∪{x}
```

以比较作为基本运算，该算法在第 2 行的排序需要 $O(m\log m)$ 时间，第 3 行的循环运行 n 次，每次做的工作量是第 5 行的二分查找，查找时间是 $O(\log m)$，因此第 3 行的 for 循环总时间是 $O(n\log m)$，于是算法时间复杂度是

$$W(n) = O(m\log m) + O(n\log m) = O((m+n)\log m)$$
$$= O(n\log m) = O(n\log\log n)$$

2.13 (1) 如果采用 3 个一组的分法，Select 算法找 m^* 的递归调用的子问题规模将变成 $n/3$，而在划分后求解第 k 小的子问题规模将至多不超过原来问题规模的 $2/3$，于是关于时间复杂度函数的递推方程变成

$$W(n) = W\left(\frac{n}{3}\right) + W\left(\frac{2n}{3}\right) + O(n)$$

利用递归树方法求解，该方程的解是 $W(n) = O(n\log n)$. 与直接排序的方法时间复杂度一样.

(2) 如果采用 7 个一组的分法，Select 算法找 m^* 的递归调用的子问题规模将变成 $n/7$，

而在划分后求解第 k 小的子问题规模将至多不超过原来问题规模的 $10/14=5/7$，于是关于时间复杂度函数的递推方程变成

$$W(n)=W\left(\frac{n}{7}\right)+W\left(\frac{5n}{7}\right)+O(n)$$

利用递归树方法求解，该方程的解是 $W(n)=O(n)$，与 5 个一组的分组方法时间复杂度一样.

2.14 算法设计思想：规定 S 的中位数 x 是从小到大排序的第 $n/2$ 个数. 用 x 划分 S，比 x 小的整数属于 S_1，x 本身也放到 S_1，其余的属于 S_2. 由于 n 是偶数，$|S_1|=|S_2|=n/2$. 易见这样的划分满足 $\left|\sum_{x\in S_1}x-\sum_{x\in S_2}x\right|$ 达到最大.

由于找中位数和划分的时间都是 $O(n)$，因此该算法的时间复杂度是 $T(n)=O(n)$.

2.15 (1) 算法 A 每次调用 Findmax 需要 $O(n)$ 时间，调用 i 次，于是

$$T_A(n)=O(in)=O(n^{3/2})$$

算法 B 排序时间是 $O(n\log n)$，输出时间是 $O(i)$，于是

$$T_B(n)=O(n\log n)+O(i)=O(n\log n)+O(n^{1/2})$$
$$=O(n\log n)$$

(2) 算法 C 的设计思想：设 k 表示第 i 大元素在从小到大排好序数组中的下标，即 $k=n-i+1$. 用选择算法确定这个元素 x；然后用 x 划分 S，将比 x 大的放到后边；排序 S 中从 k 到 n 的元素，倒序输出.

算法 C

输入：n 个数的数组 S

输出：倒序排列的前 i 个大的数

1. $k\leftarrow n-i+1$
2. $x\leftarrow \text{Select}(S,k)$
3. 采用划分算法 Partition，将 S 中比 x 大的数交换到 $S[k+1..n]$
4. 对 $S[k..n]$ 排序
5. 从后向前输出 $S[k..n]$

以比较作为基本运算，该算法的第 2 行用 $O(n)$ 时间，第 3 行用 $O(n)$ 时间，第 4 行用 $O(i\log i)$ 时间，于是总时间为

$$T_C(n)=O(n)+O(n)+O(i\log i)=O(n)+O(n^{1/2}\log n^{1/2})=O(n)$$

2.16 (1) 算法设计思想：利用找最大和最小的算法 FindMaxMin，输出 S 中最大 max 和最小 min，令 $x\leftarrow max$，$y\leftarrow min$

该算法最坏情况下的时间复杂度是 FindMaxMin 算法复杂度，即

$$W(n)=3n/2-2$$

(2) 算法设计思想：

1. 按照递增顺序排序 S 中的数
2. 令 c 是第二个数减去第一个数的差
3. 从第三个数开始，顺序计算与前一个数的差并与 c 比较
4. $n-1$ 个差中的最小差所对应的两个数就是所求的数

该算法的第一步需要 $O(n\log n)$时间,第 2 步常数时间,第 3 步需要 $O(n)$时间,因此总时间是

$$W(n)=O(n\log n)+O(1)+O(n)=O(n\log n)$$

2.17 算法设计思想：以 $s/2$ 作为标准把 L 划分成 A 与 B,使得 A 中的元素小于 $s/2$,B 中的元素大于 $s/2$. 如果存在满足要求的 x 与 y,那么必然是 A 和 B 每个集合各 1 个. 对 A 与 B 中较小的集合排序,依次处理另一集合的元素 x. 通过对排序集进行二分查找,看看是否存在 y 与 x 之和等于 s.

算法 SumEqual

输入：整数集合 L,整数 s

输出：如果存在 L 中元素 x 和 y,使得 $x+y=s$,则输出 x,y;否则输出 0

1. 用 $s/2$ 将 L 中的数划分成 A 和 B,使得 A 中元素小于 $s/2$,B 中元素大于 $s/2$
2. 令 $|A|=a$, $|B|=b$,
3. if $a>b$
4. then　按照从小到大顺序排序 B 中元素
5. 　　　对 A 中每个元素 x 利用二分查找检查在 B 中是否有元素 y,使得 $x+y=s$
6. else　按照从小到大顺序排序 A 中元素
7. 　　　对 B 中每个元素 y 利用二分查找检查在 A 中是否有元素 x,使得 $x+y=s$

假设划分后较小集合元素数为 m,算法的最坏情况的复杂度是

$$W(n)=O(n)+O(m\log m)+O(n\log m)=O(n\log m)$$

该算法在 $m\ll n$ 时是有效的,但最坏情况(当 m 与 n 的比是常数时)是 $O(n\log n)$时间的算法.

另一种可行的算法：排序 L 中的元素,然后依次处理 L 中的元素 x,用二分检索看看是否存在 y 满足 $x+y=s$. 这个算法是 $O(n\log n)$时间的算法.

2.18 算法设计思想是：先依次计算每个点到原点的距离 $d_1,d_2,\cdots,d_n$,从其中找出第 $\sqrt{n}$小的距离 d_i,依次将每个点的距离与 d_i 比较,如果小于 d_i,则是满足要求的点.

算法 FindPoint(A,B)

输入：$A=\{x_1,x_2,\cdots,x_n\}$,$B=\{y_1,y_2,\cdots,y_n\}$分别为 n 个点的横、纵坐标

输出：距离原点最近的 $\sqrt{n}$个点的标号

1. for $i\leftarrow 1$ to n
2. 　$d_i\leftarrow\sqrt{x_i^2+y_i^2}$
3. $k\leftarrow\sqrt{n}$
4. 在$\{d_1,d_2,\cdots,d_n\}$中找第 k 小的元素 b
5. 把每个 d_i 与 b 比较,把 k 个小于等于 b 的 d_i 及其下标 i 存放到 C 中
6. 对 C 中 d_i 按照从大到小顺序排序,若移动 d_i 时其下标 i 同时移动
7. for $j\leftarrow 1$ to k return $C[j]=d_i$ 的下标 i

上述算法在第 1 行和第 2 行的时间是 $O(n)$,第 4 行调用 Select 算法,时间也是 $O(n)$,第5 行的比较执行 $O(n)$ 次,每次执行用常数时间,共执行 $O(n)$时间,第 6 行的排序是 $O(k\log k)$时间,而 $k=O(\sqrt{n})$,第 7 行输出是 $O(k)$时间. 以比较做基本运算,总时间是

$$W(n) = O(n) + O(k\log k) = O(n) + O(\sqrt{n}\log\sqrt{n}) = O(n)$$

2.19 算法的设计思想：如果 A 中只有 2 个数，那么比较 1 次就可以确定最大数与最小数. 否则，将 A 划分成相等的两个子集 A_1 与 A_2. 用算法 MaxMin 递归地在 A_1 与 A_2 中找最大与最小. 令 a_1,a_2 分别表示 A_1 与 A_2 中的最大数，b_1 与 b_2 分别表示 A_1 与 A_2 中的最小数，那么 $\max\{a_1,a_2\}$ 与 $\min\{b_1,b_2\}$ 就是所需要的结果. 算法的伪码如下：

```
MaxMin(A,s,l)              //求数组 A[s..l]的最大元素 max 和最小元素 min
if s=l-1
then if A[s]<A[l] then max←A[l];min←A[s]
     else min←A[l];max←A[s].
else
     k←(s+l)/2
     MaxMin(A,s,k)
     a1←max; b1←min
     MaxMin(A,k+1,l)
     a2←max; b2←min
     if a1>a2 then max←a1
     else max←a2
     if b1<b2 then min←b1
     else min←b2
     return max,min
```

为计算给定数组，仅需调用 MaxMin$(A,1,n)$即可.

设比较次数为 $T(n)$，则

$$\begin{cases} T(n) = 2T(n/2) + 2 \\ T(2) = 1 \end{cases}$$

用 $n=2^k$ 换元得到

$$H(k) = 2H(k-1) + 2, \quad H(1) = 1$$

解得

$$H(k) = 3 \cdot 2^{k-1} - 2$$

从而得到

$$T(n) = 3n/2 - 2$$

2.20 采用芯片测试算法 Test. 将 n 个人分成 $n/2$组，每组 2 个人，如果 2 个人的回答相同，则他们或都是诚实的，或都说了谎. 如果回答不同，则至少 1 个人说谎. 回答相同的组中任意保留 1 人进入下一轮，回答不同的 2 人都淘汰. 当有轮空的人时需要单独处理. 根据芯片测试算法的分析，经过一轮测试后，人数至少减半，且进入下一轮的诚实的人数多于可能说谎的人数. 该算法的时间复杂度与 Test 算法一样，是 $O(n)$.

2.21 (1) 依次调用选择算法 Select(S,k_i)，$i=1,2,\cdots,r$. 每次调用最坏情况下的时间复杂度为 $O(n)$，总共调用 r 次，因此最坏情况下的时间复杂度为 $O(nr)$.

(2) 当 $r>1$ 时，采用分治策略，假设原始输入是(S,K)，其中 $K=\{k_1,k_2,\cdots,k_r\}$. 取 $t=r/2$，调用 Select 算法计算 S 的第 k_t 小 x. 用 x 将 S 划分成不超过 x 的数的集合 S_L 和大

于 x 的数的集合 S_R. 从而将原问题归约为(S_L,K_L)和(S_R,K_R)两个 r 减半的子问题,其中 $K_L=\{k_1,k_2,\cdots,k_{t-1}\}$,$K_R=\{k_{t+1},k_{t+2},\cdots,k_r\}$. 递归求解($S_L,K_L$)和($S_R,K_R$)即可.

设原始问题(S,K)的规模为 n 和 r,其中 $n=|S|$,$r=|K|$. 经过归约得到两个子问题,不妨设 r 是偶数,那么(S_L,K_L)的规模不超过 n_1、$r/2$,(S_R,K_R)的规模不超过 n_2、$r/2$. 令 $T(n,r)$是对(S,K)的工作量,那么有

$$T(n,r) = T(n_1,r/2) + T(n_2,r/2) + O(n) \quad r > 1$$
$$T(n,1) = O(n)$$

上述递推式中的 $O(n)$包含求 S 第 k_t 小的数 x 的工作量以及用 x 划分 S 的工作量. 用递归树求解. 不难看出,在递归树的每一层结点上的工作量之和都等于 $O(n)$,总计 $\log r$ 层,因此总工作量 $T(n,r)=O(n\log r)$.

2.22 算法如下:

1. 对 B 排序
2. 对每个 A 元素在 B 中二分检索;如果存在,则删除 B 中相等的元素
3. 将 A 中元素放到输出集合 C 中
4. 将 B 中剩余元素加到 C 集合中

该算法的时间复杂度为 $O(n\log m)=O(n\log\log n)$.

2.23 算法如下:

1. 用 Select 算法找第 $n/4$ 小的数 a 和第 $3n/4$ 小的数 b
2. if $a=b$ then return "无解"
3. else 用 a 和 b 划分数组 A 为 A_1,A_2,A_3,A_4,A_5,其中 A_1的数$<a$,A_2的数$=a$,A_3的数$>a$ 且小于 b,A_4的数$=b$. A_5的数$>b$
4. if A_3非空,则 A_3中的任意数都为近似中值,否则无解

算法的时间复杂度为 $O(n)$.

2.24 算法设计思想:对 A 排序,相同的元素是连续分布的. 通过一轮顺序比较对每种元素进行计数,同时保留个数最多的元素,最后将保留的元素输出即可. 算法时间复杂度为 $T(n)=O(n\log n)$.

2.25 算法设计思想如下:

1. 求 $x_1, x_2, \cdots, x_n$的中位数 x_m
2. 用 x_m划分数组$\{x_1, x_2, \cdots, x_n\}$,使得比 x_m小的放入集合 X_L,比 x_m大的放入 X_R
3. 计算 X_L中数的概率之和 $P(X_L)$
4. 如果 $P(X_L)=1/2$,那么输出 x_m
5. 如果 $P(X_L)<1/2$,且 $1-P(X_L)-p(x_m)\leqslant 1/2$,那么输出 x_m
6. 如果 $P(X_L)<1/2$,且 $1-P(X_L)-p(x_m)>1/2$,那么 x_k在 X_R中. 对 X_R类似地递归处理,找到它的中位数,划分 X_R,注意在计算划分元素两边数的概率之和时需对整个 X 中的数计算
7. 如果 $P(X_L)>1/2$,那么 x_k在 X_L中. 对 X_L如步骤 6 类似地递归处理

找中位数 x_m、划分数组、计算概率之和总计时间为 $O(n)$,处理后子问题规模减半,于是算法的时间复杂度 $T(n)$满足下述方程

$$T(n) = T(n/2) + O(n), \quad T(1) = 0$$

解得 $T(n)=O(n)$.

2.26 设油井的纵坐标为 $y_1, y_2, \cdots, y_n$，满足 $y_1<y_2<\cdots<y_n$，不妨设 n 是偶数. 如图 2.3 所示，假设 y 为主管线位置的纵坐标，且

$$y_{n/2} < y < y_{n/2+1}$$

即 y 的上方和下方的油井个数一样多，都是 $n/2$.

如果把 y 向下移动 Δ 距离，且有 k 个油井的位置由主管线的下方移动到上方. 那么原来在主管线上方的每个油井都增加支管线长度 Δ，总计增加长度 $(n/2)\Delta$. 原来在 $y-\Delta$ 位置下方的每个油井减少支管线长度 Δ，总计减少长度 $(n/2-k)\Delta$. 剩下的 k 个油井原来在主管线下方，现在变到主管线上方，每个减少支管线长度小于 Δ(也许还有增加支管线长度的油井)，因此减少的总量 $\delta<k\Delta$. 综合考虑以上三种情况，各支管线的长度之和增加了

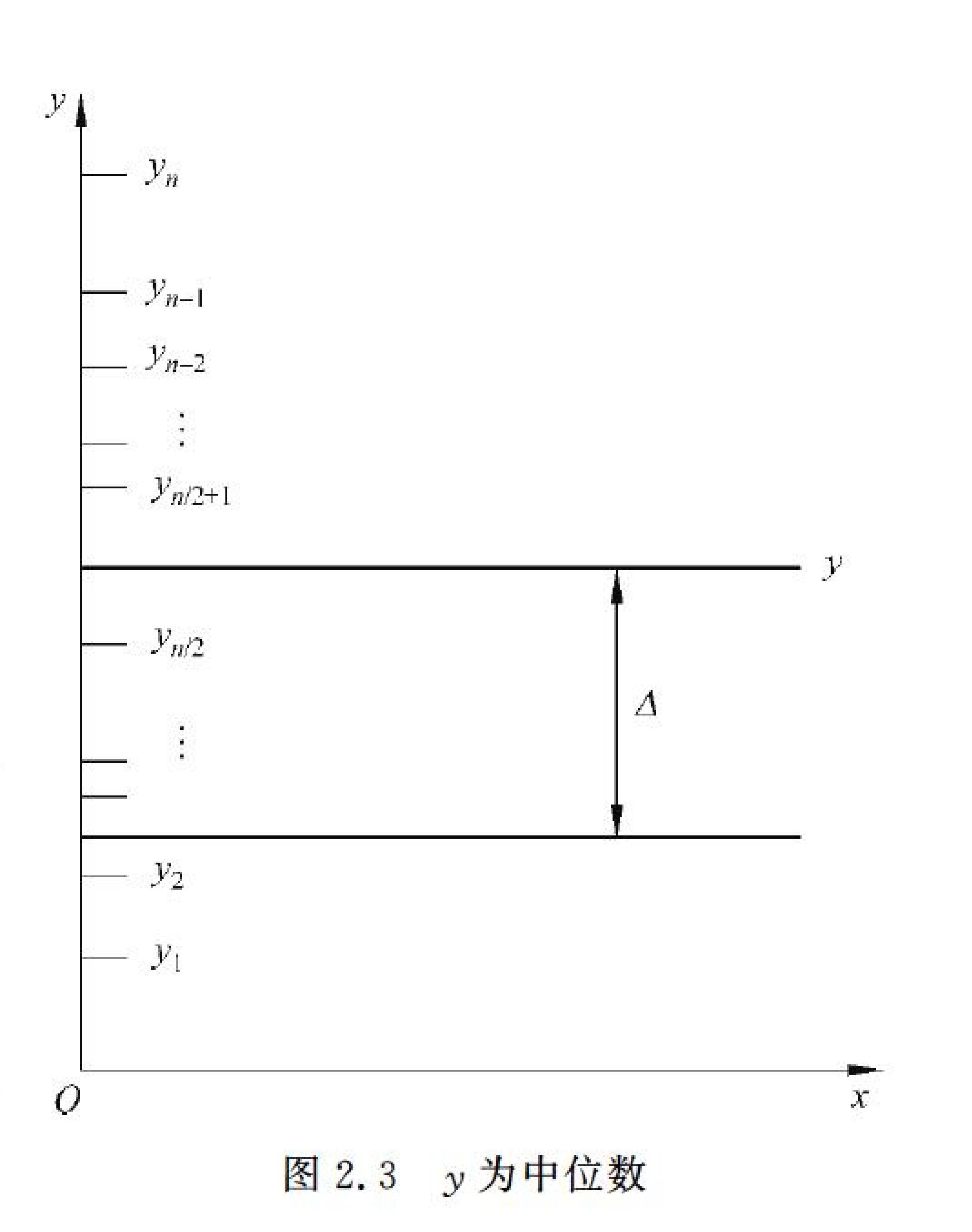


图 2.3　y 为中位数

$$(n/2)\Delta-[(n/2-k)\Delta+\delta]>(n/2)\Delta-[(n/2-k)\Delta+k\Delta]=0$$

这说明主干线从中点位置下移，只能增加支管线的总长度. 类似的分析可以证明，如果主干线从中点位置上移，同样增加支管线的总长度.

对于 n 为奇数的情况，同样可以证明，主管线的纵坐标位置 $y=y_{(n+1)/2}$ 是最佳选择.

根据上述分析，算法设计思想是：

1. 取 $y_1, y_2, \cdots, y_n$ 的中位数 y
2. 令管线纵坐标等于 y

该算法的时间复杂度是 Select 算法的运行时间，即 $O(n)$.

2.27 设 n 个商店的坐标是 $(x_1, y_1), (x_2, y_2), \cdots, (x_n, y_n)$，问题的解 (x, y) 满足

$$\min\{\,|x-x_1|+|y-y_1|+|x-x_2|+|y-y_2|+\cdots +|x-x_n|+|y-y_n|\mid 0\leqslant x, y\leqslant m\}$$

由于横、纵坐标是互相独立的，于是得到

$$\min\{|x-x_1|+|x-x_2|+\cdots+|x-x_n|\mid x=0,1,\cdots,m\}$$
$$\min\{|y-y_1|+|y-y_2|+\cdots+|y-y_n|\mid y=0,1,\cdots,m\}$$

类似于习题 2.26 石油管道选择问题的证明，只要选取 x 为 $\{x_1, x_2, \cdots, x_n\}$ 的中位数，y 为 $\{y_1, y_2, \cdots, y_n\}$ 的中位数，上述公式达到最小值.

算法设计思想：取 $x_1, x_2, \cdots, x_n$ 的中位数作为 x，$y_1, y_2, \cdots, y_n$ 的中位数作为 y 即可，算法时间复杂度为 $O(n)$.

2.28 算法的主要思想是：在二分归并排序算法中附加计数逆序的工作. 在递归调用算法分别对子数组 L_1 与 L_2 排序时，分别计数每个子数组内部的逆序；在归并排好序的子数组 L_1 与 L_2 的过程中，附带计数 L_1 的元素与 L_2 的元素之间产生的逆序. 假设 L_1 是前半个数组，L_2 是后半个数组. 如果 L_1 的最小元素 x 大于 L_2 的最小元素 y，那么算法将从 L_2 中

取走 y. 这时 L_1 中的每个元素都和 y 构成逆序,所增加的逆序数恰好等于此刻 L_1 中的元素总数. 相反,如果 L_1 的最小元素 x 小于 L_2 的最小元素 y,那么算法将从 L_1 中取走 x. 这时 L_2 中的每个元素都不会和 x 构成逆序,因此不必改变逆序总数. 在算法运行中,每次把这样增加的逆序数加到逆序总数上.

初始逆序数 $N=0$,进入递归算法,算法的主要步骤是:

1. 将数组 L 从中间划分成前后两个子数组 L_1 和 L_2
2. 递归处理 L_1
3. 递归处理 L_2
4. 在归并 L_1 与 L_2 时计数 L_1 与 L_2 的元素产生的逆序数 m,并将 m 加到 N 上

该算法的第 2 步和第 3 步是递归调用,每个子问题的规模都是原问题规模的 1/2,第 4 步每次比较至少拿走 1 个元素,因此元素之间的比较次数与二分归并排序算法的比较次数一样,是 $n-1$ 次. 如果以比较运算为基本运算,对于输入规模为 n 的数组所做的比较次数为 $T(n)$,那么有

$$\begin{cases} T(n) = 2T(n/2) + n - 1 \\ T(1) = 0 \end{cases}$$

这就是二分归并排序算法的递推方程,因此得到

$$T(n) = n\log n - n + 1$$

2.29 (1) 算法的主要步骤是:

① 用 1 个螺母与所有的螺栓尝试,将螺栓划分成比较小的子集 B_1 与比较大的子集 B_2,同时找到自己的配套螺栓 y.

② 把每个没有匹配的螺母与 y 匹配. 如果它小于 y,则把它放在集合 A_1 中;否则放在 A_2 中. A_1 中的螺母应该在 B_1 中找与之配套的螺栓;A_2 中的螺母应该在 B_2 中找与之配套的螺栓. 这样就得到两个规模为 s 和 $n-s-1$ 的子问题,其中 $s=|B_1|$.

③ 递归处理两个子问题. 当子问题规模为 1 时,不需要尝试,可直接匹配.

(2) 递推方程是

$$\begin{cases} T(n) = T(s) + T(n-1-s) + 2n - 1 \\ T(1) = 0 \end{cases}$$

该方程与快速排序算法的方程类似,最坏情况下为 $W(n)=O(n^2)$,平均情况下为 $A(n)=O(n\log n)$.

2.30 (1) 顺序从下到上测试,一次一个高度,最坏情况下时间 $W(n)=O(n)$.

(2) 二分查找算法,取 $n/2$ 高度进行第一次测试,如果瓶子没有摔碎,那么它的强度位于 $n/2+1$ 到 n 之间;如果摔碎了,那么它的强度位于 1 到 $n/2-1$ 之间. 每次测试需要一个瓶子的代价,测试后查找的范围减半,最坏情况下的时间复杂度 $W(n)=O(\log n)$.

(3) 为简单起见,不妨设 $\sqrt{n}$ 为整数. 将高度 $1,2,\cdots,n$ 分成 $\sqrt{n}$ 个组,每组 $\sqrt{n}$ 个高度,即第 j 组含有的高度为

$$(j-1)\sqrt{n}+1,(j-1)\sqrt{n}+2,\cdots,(j-1)\sqrt{n}+\sqrt{n}, \quad j=1,2,\cdots,\sqrt{n}$$

取第一个瓶子,从下到上顺序测试每组的最大高度,即高度 $\sqrt{n},2\sqrt{n},\cdots,n$. 如果前 $j-1$ 组的检查中瓶子都没有摔碎,而在第 j 组检查时瓶子摔碎了,那么玻璃瓶的强度处在第 j 组内

的$\sqrt{n}$个高度值之中. 于是,至多经过$\sqrt{n}$次测试,待检查的高度范围就缩减到原来的$\frac{1}{\sqrt{n}}$. 接着取第二个瓶子,从下到上顺序测试第 j 组的$\sqrt{n}$ 个高度,至多经过$\sqrt{n}$ 次测试,就可以得到瓶子的强度. 测试次数满足

$$W(n) = O(\sqrt{n}) + O(\sqrt{n}) = O(\sqrt{n})$$

(4) 和(3)的情况类似,因为瓶子有 k 个,我们希望每个瓶子的测试工作量尽可能均衡,都是 $n^{1/k}$. 每次分组的组数都是 $n^{1/k}$. 为简单起见,这里不妨设 $n^{1/k}$为整数.

初始,对于给定的 k,将高度 $1,2,\cdots,n$ 分成 $n^{1/k}$组,每组 $n^{(k-1)/k}$个元素. 用第一个瓶子从下到上顺序测试每组的最大高度,至多 $n^{1/k}$次测试,就可以确定瓶子强度所在的组,测试规模缩小为 $n^{(k-1)/k}$,剩下 $k-1$ 个瓶子样品,测试时间 $W_1(n)=O(n^{1/k})$.

接着,将 $n^{(k-1)/k}$高度仍旧分成 $n^{1/k}$个组,每组 $n^{(k-2)/k}$个高度. 取第二个样品进行类似的测试,至多测试 $n^{1/k}$次就可以确定强度所在的分组,测试规模缩小为 $n^{(k-2)/k}$,剩下 $k-2$ 个瓶子样品,时间 $W_2(n)=O(n^{1/k})$.

如此继续,每轮测试的次数都是 $n^{1/k}$,所含高度数不断减少,依次为 $n^{(k-1)/k}$,$n^{(k-2)/k}$,$n^{(k-3)/k}$……直到第 k 个样品(最后一个样品)时,待测高度只剩下 $n^{1/k}$个. 于是经过从下到上至多 $n^{1/k}$次测试就可以确定瓶子的强度. 总的测试时间为

$$W(n) = W_1(n) + W_2(n) + \cdots + W_k(n) = kO(n^{1/k}) = O(kn^{1/k})$$

第 3 章

动态规划

3.1 内容提要

1. 基本概念

动态规划(dynamic programming)是一种求解多阶段决策(优化)问题的算法设计技术,其主要思想是:将原问题归约为规模较小、结构相同的子问题,建立原问题与子问题优化函数间的依赖关系. 从规模最小的子问题开始,利用上述依赖关系求解规模更大的子问题,直到得到原始问题的解为止.

动态规划算法的适用条件 适用于求解多阶段决策(优化)问题,该问题的解可以表示为一个决策序列,且满足优化原则(或最优子结构性质),即一个最优决策序列的任何子序列本身一定是相对于子序列的初始和结束状态的最优决策序列.

对于规模大的实例,对存储空间的需求是该算法的瓶颈.

主要设计步骤

1. 写出所要求解的组合优化问题的目标函数和约束条件.

2. 确定子问题的结构与边界,将问题求解转变成多步判断的过程.

3. 定义优化函数,以该函数的极大值或者极小值作为判断的依据,确定是否满足优化原则.

4. 列出有关优化函数的递推关系和边界条件.

5. 根据问题的解的不同情况,考虑是否需要设立标记函数.

6. 从初值开始,自底向上计算每个子问题的优化函数值(有的需要同时计算标记函数),并以备忘录的方式存储所有的中间结果.

7. 如果有标记函数,则根据标记函数逐步追踪问题的解.

2. 时间复杂度的分析方法

时间复杂度取决于备忘录(包括优化函数及标记函数)中每个项的计算工作量,以及用标记函数追踪解的工作量. 大多数情况下,追踪解的工作量不超过优化函数计算的工作量,因此可以根据递推关系确定备忘录中每个项的计算工作量,然后对这些工作量求和.

3. 典型的动态规划算法

矩阵链相乘 MatrixChain(P,n) 给定矩阵链 $A_{1..n}=A_1A_2\cdots A_n$,其行和列数为向量 $P=<P_0,P_1,\cdots,P_n>$,其中 P_0 是 A_1 的行数,$P_i(i=1,2,\cdots,n-1)$是 A_i 的列数和 A_{i+1} 的行数,P_n 是 A_n 的列数. 通过加括号确定具有最少乘法次数的运算顺序.

设子问题 $A_{i..j}$ 是矩阵链 $A_iA_{i+1}\cdots A_j$ 的计算，令 $m[i,j]$ 是计算 $A_{i..j}$ 的最少的乘法次数，则

$$m[i,j]=\begin{cases}0 & i=j\\ \min\limits_{i\leqslant k<j}\{m[i,k]+m[k+1,j]+P_{i-1}P_kP_j\} & i<j\end{cases},\ 1\leqslant i\leqslant j\leqslant n$$

用标记函数 $s[i,j]$ 记录使得优化函数 $m[i,j]$ 达到最小值时的 k 值，可用于追踪加括号的位置，备忘录是 $n\times n$ 的 $m[i,j]$ 表和 $s[i,j]$ 表，$i,j=1,2,\cdots,n$，时间是 $O(n^3)$.

背包问题(knapsack problem)　一个旅行者准备随身携带一个背包. 可以放入背包的物品有 n 种，每种物品的重量和价值分别为 w_j 和 v_j，$j=1,2,\cdots,n$. 如果每种物品放入背包的数量不限，背包的最大重量限制是 b，怎样选择放入背包的物品以使得背包的价值最大？

使用组合优化问题的描述方法，设 x_j 表示装入背包的第 j 种物品的数量，背包问题可以描述为

$$\text{目标函数}\quad \max\sum_{j=1}^{n}v_jx_j$$

$$\text{约束条件}\quad \sum_{j=1}^{n}w_jx_j\leqslant b$$

$$x_j\in\mathbf{N},\quad j=1,2,\cdots,n$$

当每种物品只有 1 个，即 x_j 只能取 0 或 1 时，称为 **0-1 背包问题.**

使用动态规划的方法求解背包问题. 设 $F_k(y)$ 表示只允许装前 k 种物品，背包总重不超过 y 时背包的最大价值，则

$$F_k(y)=\max\{F_{k-1}(y),F_k(y-w_k)+v_k\},\ k=1,2,\cdots,n,\ y=1,2,\cdots,b$$

$$F_0(y)=0\qquad 0\leqslant y\leqslant b$$

$$F_k(0)=0\qquad 0\leqslant k\leqslant n$$

$$F_k(y)=-\infty\qquad y<0$$

根据上面的递推式，当 $k=1$ 时，可以得到

$$F_1(y)=\left\lfloor\frac{y}{w_1}\right\rfloor v_1$$

令 $i_k(y)$ 表示在计算优化函数值 $F_k(y)$ 时所用到物品的最大标号，则

$$i_k(y)=\begin{cases}i_{k-1}(y) & F_{k-1}(y)>F_k(y-w_k)+v_k\\ k & F_{k-1}(y)\leqslant F_k(y-w_k)+v_k\end{cases},\ 1<k\leqslant n,\ y=1,2,\cdots,b$$

$$i_1(y)=\begin{cases}1 & y\geqslant w_1\\ 0 & y<w_1\end{cases}$$

该算法最坏情况下的时间复杂度为 $O(nb)$.

最长公共子序列 LCS(X,Y,m,n)　给定长度分别为 m 和 n 的序列 X 和 Y，求 X 与 Y 的最长公共子序列 Z.

设 $X_i=<x_1,x_2,\cdots,x_i>$，$Y_j=<y_1,y_2,\cdots,y_j>$，设 $C[i,j]$ 表示 X_i 与 Y_j 的最长公共子序列的长度，则

$$C[i,j]=\begin{cases}C[i-1,j-1]+1 & \text{如果 } i,j>0,\ x_i=y_j\\ \max\{C[i,j-1],C[i-1,j]\} & \text{如果 } i,j>0,\ x_i\neq y_j\end{cases}$$

$$i = 1,2,\cdots,m, \quad j = 1,2,\cdots,n$$
$$C[0,j] = C[i,0] = 0, \quad 对一切\ 1 \leqslant i \leqslant m,\ 1 \leqslant j \leqslant n$$

令 $B[i,j]$为标记函数. 在归约为子问题时,如果 $C[i,j]$归约为 $C[i-1,j-1]$,则 $B[i,j]$="↖";若归约为 $C[i,j-1]$,则 $B[i,j]$="←";若归约为 $C[i-1,j]$,则 $B[i,j]$="↑". 算法最坏情况下的时间复杂度为 $O(mn)$.

图像压缩 Compress(P,n)　给定图像的灰度值序列 $P=\langle p_1,p_2,\cdots,p_n\rangle$,其中 p_i 为第i 个像素的灰度值,采用变位压缩存储格式,如何对 P 进行分段,以使得存储 P 占用的二进制位数达到最少?

设 $S[i]$表示输入为$\langle p_1,p_2,\cdots,p_i\rangle$时按最优分段存储所使用的二进制位数,则

$$S[i] = \min_{1\leqslant j\leqslant \min\{i,256\}} \{S[i-j] + j * b[i-j+1,i] + 11\} \quad i = 2,3,\cdots,n$$
$$S[1] = b[1,1] + 11$$

其中 $b[i-j+1,i]$表示最后一段$\langle p_{i-j+1},\cdots,p_i\rangle$中的最大灰度值在存储时所占用的二进制位数. 11 是存储该分段信息的额外开销. 用标记函数 $l[i]$记录进行最优划分时最后一段的像素个数,可用于追踪达到最小存储位数时的分段位置. 该算法最坏情况下的时间复杂度为 $O(n)$.

最大子段和 MaxSum(A,n)　给定 n 个整数的序列 $A=\langle a_1,a_2,\cdots,a_n\rangle$,求

$$\max\left\{0, \max_{1\leqslant i\leqslant j\leqslant n}\sum_{k=i}^{j} a_k\right\}$$

定义 $C[i]$是输入 $A[1..i]$中必须包含元素 $A[i]$的最大子段和,最优解为 OPT(A),则

$$C[i] = \max\{C[i-1] + A[i], A[i]\} \quad i = 2,\cdots,n$$
$$C[1] = A[1]$$
$$\mathrm{OPT}(A) = \max_{1\leqslant i\leqslant n}\{C[i]\}$$

该算法的时间复杂度是 $O(n)$.

最优二分检索树　设 $S=\langle x_1,x_2,\cdots,x_n\rangle$是数据集,称$(-\infty,x_1),(x_1,x_2),\cdots,(x_{n-1},x_n),(x_n,+\infty)$为第 $0,1,2,\cdots,n$ 个空隙. 与 S 相关的存取概率分布是 $P=\langle a_0,b_1,a_1,b_2,\cdots,a_i,b_{i+1},\cdots,b_n,a_n\rangle$,其中 x 处在 x_i 的概率是 $b_i(i=1,2,\cdots,n)$,处在第 j 个空隙的概率是 $a_j(j=0,1,\cdots,n)$. 求一棵关于 S 的二分检索树,使得平均比较次数

$$t = \sum_{i=1}^{n} b_i(1 + d(x_i)) + \sum_{j=0}^{n} a_j d(L_j)$$

达到最小,其中 $d(x_i)$表示树中数据结点 x_i 的深度,$d(L_j)$表示树中空隙结点 L_j 的深度.

令 $S[i,j]=\langle x_i,x_{i+1},\cdots,x_j\rangle$表示 S 以 i 和 j 作为边界的子数据集,$P[i,j]=\langle a_{i-1},b_i,a_{i+1},b_{i+1},\cdots,b_j,a_{j+1}\rangle$是子数据集 $S[i,j]$所对应的存取概率分布,$P[i,j]$与 $S[i,j]$一起构成以 i 和 j 作为边界的子问题. $m[i,j]$是针对这个子问题的最优二分检索树的平均比较次数,则

$$m[i,j] = \min_{i\leqslant k\leqslant j}\{m[i,k-1] + m[k+1,j] + w[i,j]\} \quad 1 \leqslant i \leqslant j \leqslant n$$
$$m[i,i-1] = 0 \quad i = 1,2,\cdots,n$$

其中

$$w[i,j] = \sum_{p=i-1}^{j} a_p + \sum_{q=i}^{j} b_q$$

是 $P[i,j]$ 中所有元素(包括数据元素与空隙)的概率之和. 算法最坏情况下的时间复杂度是 $O(n^3)$.

3.2 习　　题

对于本章与算法设计有关的习题,解题要求如下：必须对优化函数和标记函数的含义给出说明,列出子问题计算中所使用关于优化函数及标记函数的递推关系和初值. 根据题目要求给出迭代实现的伪码并估计算法的复杂度. 如果题目给出具体的输入实例,应该根据给定实例进行计算并给出相关的备忘录表和最后的解.

3.1 用动态规划算法求解下面的组合优化问题,

$$\max \quad g_1(x_1)+g_2(x_2)+g_3(x_3)$$
$$x_1^2+x_2^2+x_3^2 \leqslant 10$$
$$x_1,x_2,x_3 \text{ 为非负整数}$$

其中函数 $g_1(x)$,$g_2(x)$,$g_3(x)$的值给在表 3.1 中.

表 3.1　函数值

x	$g_1(x)$	$g_2(x)$	$g_3(x)$	x	$g_1(x)$	$g_2(x)$	$g_3(x)$
0	2	5	8	2	7	16	17
1	4	10	12	3	11	20	22

3.2 设 $A=\langle x_1,x_2,\cdots,x_n\rangle$ 是 n 个不等的整数构成的序列,A 的一个单调递增子序列是序列 $\langle x_{i_1},x_{i_2},\cdots,x_{i_k}\rangle$,使得 $i_1<i_2<\cdots<i_k$,且 $x_{i_1}<x_{i_2}<\cdots<x_{i_k}$. 子序列 $\langle x_{i_1},x_{i_2},\cdots,x_{i_k}\rangle$ 的长度是含有的整数个数 k. 例如 $A=\langle 1,5,3,8,10,6,4,9\rangle$,它的长为 4 的递增子序列是：$\langle 1,5,8,10\rangle$,$\langle 1,5,8,9\rangle$,…. 设计一个算法求 A 的一个最长的单调递增子序列,分析算法的时间复杂度. 设算法的输入实例是 $A=\langle 2,8,4,-4,5,9,11\rangle$,给出算法的计算过程和最后的解.

3.3 有 n 个底面为长方形的货柜需要租用库房存放. 如果每个货柜都必须放在地面上,且所有货柜的底面宽度都等于库房的宽度,那么第 i 个货柜占用库房面积大小只需要用它的底面长度 l_i 来表示,$i=1,2,\cdots,n$. 设库房总长度是 $D\left(l_i\leqslant D \text{ 且} \sum_{i=1}^{n} l_i>D\right)$. 设第 i 号货柜的仓储收益是 v_i,若要求库房出租的收益达到最大,问如何选择放入库房的货柜? 若 $l_1,l_2,\cdots,l_n,D$ 都是正整数,设计一个算法求解这个问题,给出算法的伪码描述并估计算法最坏情况下的时间复杂度.

3.4 设有 n 项任务,加工时间分别表示为正整数 $t_1,t_2,\cdots,t_n$. 现有 2 台同样的机器,从 0 时刻开始安排对这些任务的加工. 规定只要有待加工的任务,任何机器就不得闲置. 如果直到时刻 T 所有任务都完成了,总的加工时间就等于 T. 设计一个算法找到使得总加工时间 T 达到最小的调度方案. 设给定实例如下：

$$t_1=1,\quad t_2=5,\quad t_3=2,\quad t_4=10,\quad t_5=3$$

试给出一个加工时间最少的调度方案. 给出计算过程和问题的解.

3.5 设有 n 种不同面值的硬币,第 i 种硬币的币值是 v_k(其中 $v_1=1$),重量是 w_i,$i=1,2,\cdots,n$ 且现在购买某些总价值为 y 的商品,需要用这些硬币付款,如果每种钱币使用的个数不限,那么如何选择付款的方法使得付出钱币的总重量最轻?设计一个求解该问题的算法,给出算法的伪码描述并分析算法的时间复杂度. 假设问题的输入实例是:

$$v_1=1,\quad v_2=4,\quad v_3=6,\quad v_4=8$$
$$w_1=1,\quad w_2=2,\quad w_3=4,\quad w_4=6$$
$$y=12$$

给出算法在该实例上计算的备忘录表和标记函数表,并说明付线的方法.

3.6 n 种币值 $x_1,x_2,\cdots,x_n$ 和总钱数 M 都是正整数. 如果每种币值的钱币至多使用1次,问:对于 M 是否可以有一种找零钱的方法?设计一个算法求解上述问题. 说明算法的设计思想,分析算法最坏情况下的时间复杂度.

3.7 在一条直线的公路两旁有 n 个位置 $x_1,x_2,\cdots,x_n$ 可以开商店,在位置 x_i 开商店的预期收益是 p_i,$i=1,2,\cdots,n$. 如果任何两个商店之间的距离必须至少为 d 千米,那么如何选择开设商店的位置使得总收益达到最大?

(1) 用组合最优化方法对该问题建模,写出目标函数与约束条件.

(2) 设计一个算法求解该问题,说明算法设计思想,分析算法最坏情况下的时间复杂度.

3.8 设 $A=\{a_1,a_2,\cdots,a_n\}$ 是正整数的集合,且 $\sum_{i=1}^{n}a_i=N$,设计一个算法判断是否能够把 A 划分成两个子集 A_1 和 A_2,使得 A_1 中的数之和与 A_2 中的数之和相等?说明算法的设计思想,估计算法最坏情况下的时间复杂度.

3.9 有 n 项作业的集合 $J=\{1,2,\cdots,n\}$,每项作业 i 有加工时间 $t(i)\in\mathbf{Z}^+$,$t(1)\leqslant t(2)\leqslant\cdots\leqslant t(n)$,效益值 $v(i)$,任务的结束时间 $D\in\mathbf{Z}^+$,其中 $\mathbf{Z}^+$ 表示正整数集合. 一个可行调度是对 J 的子集 A 中任务的一个安排,对于 $i\in A$,$f(i)$ 是开始时间,且满足下述条件:

$$f(i)+t(i)\leqslant f(j)\text{ 或者 }f(j)+t(j)\leqslant f(i),\ j\neq i,\ i,j\in A$$
$$\sum_{k\in A}t(k)\leqslant D$$

设机器从0时刻开动,只要有作业就不闲置,求具有最大总效益的调度. 给出算法的伪码,分析算法的时间复杂度.

3.10 把0-1背包问题加以推广. 设有 n 种物品,第 i 种物品的价值是 v_i,重量是 w_i,体积是 c_i,且装入背包的重量限制是 W,体积是 V. 问如何选择装入背包的物品使得其总重不超过 W,总体积不超过 V 且价值达到最大?设计一个动态规划算法求解这个问题,说明算法的时间复杂度.

3.11 有 n 个分别排好序的整数数组 $A_0,A_1,\cdots,A_{n-1}$,其中 A_i 含有 x_i 个整数,$i=0,1,\cdots,n-1$. 已知这些数组顺序存放在一个圆环上,现在要将这些数组合并成一个排好序的大数组,且每次只能把两个在圆环上处于相邻位置的数组合并. 问如何选择这 $n-1$ 次合并的次序以使得合并时在最坏情况下总的比较次数达到最少?设计一个动态规划算法求解这个问题,说明算法的时间复杂度.

3.12 设 A 是顶点为 $1,2,\cdots,n$ 的凸多边形,可以用不在内部相交的 $n-3$ 条对角线将

A 划分成三角形，图 3.1 就是五边形的所有的划分方案. 假设凸 n 边形的边及对角线的长度 d_{ij} 都是给定的正整数，$1 \leqslant i < j \leqslant n$. 划分后三角形 ijk 的权值等于其周长，求具有最小权值的划分方案. 设计一个动态规划算法求解这个问题，说明算法的时间复杂度.

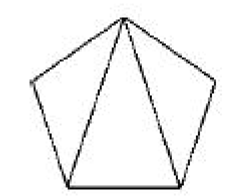
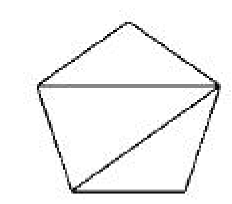
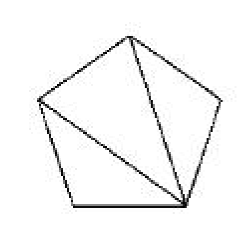
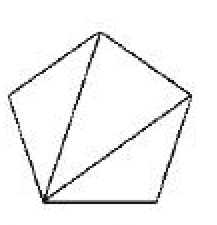
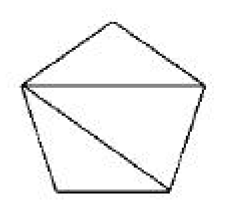
    

图 3.1　五边形的划分方案

3.13　图的连通性问题是图论研究的重要问题之一，在实际中有着广泛的应用. 例如通信网络的连通问题，运输路线的规划问题等. 一个著名的检查图的连通性的算法就是 Warshall 算法. 假设 $D = \langle V, E \rangle$ 是顶点集为 $V = \{x_1, x_2, \cdots, x_n\}$，边集为 E 的有向图. $n \times n$ 的 0-1 矩阵 $M = (r_{ij})$ 是 D 的矩阵表示. 考虑 $n+1$ 个矩阵构成的序列 $M_0, M_1, \cdots, M_n$，将矩阵 M_k 的 i 行 j 列的元素记作 $M_k[i,j]$. 对于 $k=0,1,\cdots,n$，$M_k[i,j]=1$ 当且仅当在图中存在一条从 x_i 到 x_j 的路径，并且这条路径除端点外中间只经过 $\{x_1, x_2, \cdots, x_k\}$ 中的顶点. 不难看出 M_0 就是 M，而在 M_n 中如果 $M_n[i,j]=1$，则说明 D 中 x_i 与 x_j 是连通的. Warshall 算法从 M_0 开始，顺序计算 $M_1, M_2, \cdots$，直到 M_n 为止. 利用动态规划的迭代实现方法来实现 Warshall 算法，给出算法的伪码表示并分析算法的时间复杂度. 假设某有向网络的结点是 a, b, c, d，已知网络的矩阵表示是

$$M = \begin{bmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

给出算法在这个实例的计算过程和结果.

3.14　考虑 3.3.7 节中提到的带权编辑距离问题. 设 $S_1[1..n]$ 和 $S_2[1..m]$ 表示两个序列，假设插入和删除操作的权是 d，替换操作的权是 r. 令 $C[i,j]$ 表示序列 $S_1[1..i]$ 和 $S_2[1..j]$ 的最小权编辑距离，设计一个算法求解该问题，给出关于 $C[i,j]$ 的递推关系并分析算法的时间复杂度.

3.15　某机器每天接受大量加工任务，第 i 天需要加工的任务数是 x_i. 随着机器连续运行时间的增加，处理能力越来越低，需要花 1 天时间对机器进行检修，以提高处理能力. 检修当天必须停工，重启后的第 i 天能够加工的任务数是 s_i，且满足

$$s_1 > s_2 > \cdots > s_n > 0$$

我们的问题是：给定 $x_1, x_2, \cdots, x_n, s_1, s_2, \cdots, s_n$，如何安排机器的检修时间，以使得在 n 天内加工的任务数达到最大？设计一个算法求解该问题.

3.16　设 P 是一台 Internet 上的 Web 服务器. $T = \{1, 2, \cdots, n\}$ 是 n 个下载请求的集合，$\forall i \in T$，$a_i \in \mathbf{Z}^+$ 表示下载请求 i 所申请的带宽. 已知服务器的最大带宽是正整数 K. 我们的目标是使带宽得到最大限度的利用，即确定 T 的一个子集 S，使得 $\sum_{i \in S} a_i \leqslant K$，且 $K - \sum_{i \in S} a_i$ 的值达到最小. 设计一个算法求解服务器下载问题，用文字说明算法的主要设计思想和步骤，给出最坏情况下的时间复杂度.

3.17 有正实数构成的数字三角形排列形式如图3.2所示. 第一行的数为 a_{11};第二行的数从左到右依次为 a_{21},a_{22};以此类推,第 n 行的数为 $a_{n1},a_{n2},\cdots,a_{nn}$. 从 a_{11} 开始,每一行的数 a_{ij} 只有两条边可以分别通向下一行的两个数 $a_{(i+1)j}$ 和 $a_{(i+1)(j+1)}$. 请设计一个算法,计算出从 a_{11} 通到 $a_{n1},a_{n2},\cdots,a_{nn}$ 中某个数的一条路径,并且使得该路径上的数之和达到最小.

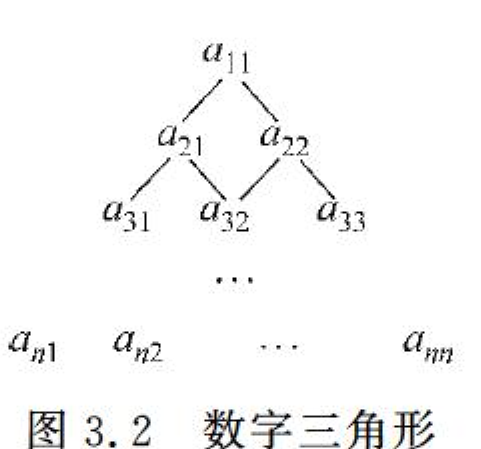


图3.2 数字三角形

3.18 设集合 $A=\{a,b,c\}$,A 中元素的运算满足 $aa=ab=bb=b,ac=bc=ca=a,ba=cb=cc=c$. 给定 A 中 n 个字符组成的串 $x_1x_2\cdots x_n$,设计一个算法,检查是否存在一种运算顺序使得这个串的运算结果等于 a. 如果存在,则回答"1",否则回答"0". 例如串 $x=bbbba$,因为存在下述运算顺序,使得

$$(b(bb))(ba) = (bb)(ba) = b(ba) = bc = a$$

因此回答"1". 而对串 bca,由于 $(bc)a=aa=b,b(ca)=ba=c$,因此回答"0". 说明算法的设计思想,并给出最坏情况下的时间复杂度.

3.3 习题解答与分析

3.1 设 $F_k(y)$ 表示对 $x_1,x_2,\cdots,x_k$ 赋值,且 $x_1^2+x_2^2+\cdots+x_k^2\leqslant y$ 时所得到的目标函数的最大值,递推关系和初值是

$$F_k(y) = \max_{0\leqslant x_k\leqslant\sqrt{y}}\{F_{k-1}(y-x_k^2)+g_k(x_k)\}$$

$$F_1(y) = g_1(\sqrt{y})$$

针对给定实例的计算过程如下.

$k=1$:

$F_1(0)=2$ $\quad x_1=0$

$F_1(1)=F_1(2)=F_1(3)=g_1(1)=4$ $\quad x_1=1$

$F_1(4)=F_1(5)=F_1(6)=F_1(7)=F_1(8)=g_1(2)=7$ $\quad x_1=2$

$F_1(9)=F_1(10)=g_1(3)=11$ $\quad x_1=3$

$k=2$:

$F_2(0)=\max\{F_1(0)+g_2(0)\}=7$ $\quad x_2=0$

$F_2(1)=\max\{F_1(1)+g_2(0),F_1(0)+g_2(1)\}=12$ $\quad x_2=1$

$F_2(2)=\max\{F_1(2)+g_2(0),F_1(1)+g_2(1)\}=14$ $\quad x_2=1$

$F_2(3)=\max\{F_1(3)+g_2(0),F_1(2)+g_2(1)\}=14$ $\quad x_2=1$

$F_2(4)=\max\{F_1(4)+g_2(0),F_1(3)+g_2(1),F_1(0)+g_2(2)\}=18$ $\quad x_2=2$

$F_2(5)=\max\{F_1(5)+g_2(0),F_1(4)+g_2(1),F_1(1)+g_2(2)\}=20$ $\quad x_2=2$

$F_2(6)=\max\{F_1(6)+g_2(0),F_1(5)+g_2(1),F_1(2)+g_2(2)\}=20$ $\quad x_2=2$

$F_2(7)=\max\{F_1(7)+g_2(0),F_1(6)+g_2(1),F_1(3)+g_2(2)\}=20$ $\quad x_2=2$

$F_2(8)=\max\{F_1(8)+g_2(0),F_1(7)+g_2(1),F_1(4)+g_2(2)\}=23$ $\quad x_2=2$

$F_2(9)=\max\{F_1(9)+g_2(0),F_1(8)+g_2(1),F_1(5)+g_2(2),F_1(0)+g_2(3)\}=23$ $\quad x_2=2$

$F_2(10)=\max\{F_1(10)+g_2(0),F_1(9)+g_2(1),F_1(6)+g_2(2),F_1(1)+g_2(3)\}=24$ $\quad x_2=3$

$k=3$:

$F_3(0)=\max\{F_2(0)+g_3(0)\}=15$ $x_3=0$

$F_3(1)=\max\{F_2(1)+g_3(0),F_2(0)+g_3(1)\}=20$ $x_3=0$

$F_3(2)=\max\{F_2(2)+g_3(0),F_2(1)+g_3(1)\}=24$ $x_3=1$

$F_3(3)=\max\{F_2(3)+g_3(0),F_2(2)+g_3(1)\}=26$ $x_3=1$

$F_3(4)=\max\{F_2(4)+g_3(0),F_2(3)+g_3(1),F_2(0)+g_3(2)\}=26$ $x_3=1$

$F_3(5)=\max\{F_2(5)+g_3(0),F_2(4)+g_3(1),F_2(1)+g_3(2)\}=30$ $x_3=1$

$F_3(6)=\max\{F_2(6)+g_3(0),F_2(5)+g_3(1),F_2(2)+g_3(2)\}=32$ $x_3=1$

$F_3(7)=\max\{F_2(7)+g_3(0),F_2(6)+g_3(1),F_2(3)+g_3(2)\}=32$ $x_3=1$

$F_3(8)=\max\{F_2(8)+g_3(0),F_2(7)+g_3(1),F_2(4)+g_3(2)\}=35$ $x_3=2$

$F_3(9)=\max\{F_2(9)+g_3(0),F_2(8)+g_3(1),F_2(5)+g_3(2),F_2(0)+g_3(3)\}=37$ $x_3=2$

$F_3(10)=\max\{F_2(10)+g_3(0),F_2(9)+g_3(1),F_2(6)+g_3(2),F_2(1)+g_3(3)\}=37$ $x_3=2$

关于中间结果的备忘录如表 3.2 所示.

表 3.2　备忘录

y	$k=1$		$k=2$		$k=3$	
	$F_1(y)$	x_1	$F_2(y)$	x_2	$F_3(y)$	x_3
0	2	0	7	0	15	0
1	4	1	12	1	20	0
2	4	1	14	1	24	1
3	4	1	14	1	26	1
4	7	2	18	2	26	1
5	7	2	20	2	30	1
6	7	2	20	2	32	1
7	7	2	20	2	32	1
8	7	2	23	2	35	2
9	11	3	23	2	37	2
10	11	3	24	3	37	2

从而得到,$F_3(10)=37$,此刻 $x_3=2$,于是

$$x_1^2+x_2^2\leqslant 10-2^2=6$$

再查 $F_2(6)=20$,此刻 $x_2=2$,于是

$$x_1^2\leqslant 6-2^2=2$$

再查 $F_1(2)=4$ 得 $x_1=1$. 问题的解是:在 $x_1=1,x_2=2,x_3=2$ 时得到 $g_1(x_1)+g_2(x_2)+g_3(x_3)$的最大值 37.

3.2　使用动态规划设计技术. 对于 $i=1,2,\cdots,n$,考虑以 x_i 作为最后项的最长递增子序列的长度 $C[i]$. 如果在 x_i 项前面存在 $x_j<x_i$,那么 $C[i]=\max\{C[j]\}+1$;否则 $C[i]=1$. 因此

$$C[i]=\begin{cases}\max\{C[j]\}+1 & \exists j(1\leqslant j<i,x_j<x_i)\\ 1 & \forall j(1\leqslant j<i,x_j>x_i)\end{cases},\ i>1$$

$$C[1] = 1$$

在计算 $C[i]$ 时,用 $k[i]$ 记录 $C[i]$ 取得最大值时的 j 的值;如果不存在这样的 j,令 $k[i]=0$. 这个记录用于追踪解. 所求的最长递增子序列的长度是

$$C = \max\{C[i] \mid i = 1,2,\cdots,n\}$$

对每个 i,需要检索比 i 小的所有的 j,需要 $O(n)$ 时间,i 的取值有 n 种,于是算法时间复杂度是 $W(n)=O(n^2)$.

对于给定的实例 $A=\langle 2,8,4,-4,5,9,11\rangle$,具体的计算过程如下:

$C[1] = 1$

$C[2] = \max\{C[1]+1\} = 2$ $\qquad k[2] = 1$

$C[3] = \max\{C[1]+1\} = 2$ $\qquad k[3] = 1$

$C[4] = 1$ $\qquad k[4] = 0$

$C[5] = \max\{C[1]+1, C[3]+1, C[4]+1\} = 3$ $\qquad k[5] = 3$

$C[6] = \max\{C[1]+1, C[2]+1, C[3]+1, C[4]+1, C[5]+1\} = 4$ $\qquad k[6] = 5$

$C[7] = \max\{C[1]+1, C[2]+1, C[3]+1, C[4]+1, C[5]+1, C[6]+1\} = 5$ $\qquad k[7] = 6$

在 $C[1],C[2],\cdots,C[7]$ 中,$C[7]=5$ 是最大值,这意味着 A 的第 7 项 $x_7=11$ 是最长递增子序列的最后项,长度是 5. 子序列的构造从后向前进行,开始是 11,追踪过程是:

$k[7] = 6 \Rightarrow x_6 = 9$, $\quad k[6] = 5 \Rightarrow x_5 = 5$, $\quad k[5] = 3 \Rightarrow x_3 = 4$, $\quad k[3] = 1 \Rightarrow x_1 = 2$

于是得到解是:$\langle 2,4,5,9,11\rangle$,长度是 5. 本题可以有多个解.

3.3 类似于 0-1 背包问题,库房的长度相当于背包的重量限制,每个货柜的收益相当于物品的价值. 于是问题是:

$$\max \sum_{i=1}^{n} v_i x_i$$

$$\sum_{i=1}^{n} l_i x_i \leqslant D, \quad x_i = 0,1$$

令 $C[k,y]$ 是只允许装前 k 个货柜,库房长度为 y 时的最大收益,那么有

$$C[k,y] = \begin{cases} C[k-1,y] & y < l_k \\ \max\{C[k-1,y], C[k-1,y-l_k]+v_k\} & D \geqslant y \geqslant l_k \end{cases}, \quad k > 1$$

$$C[1,y] = \begin{cases} v_1 & y \geqslant l_1 \\ 0 & y < l_1 \end{cases}$$

算法的伪码是:

```
Store
输入:数组 L[1..n],V[1..n],D      //L 和 V 是货柜长度和价值序列,D 为库房长度
输出:最大的收益 C[n,D]
1. for y←1 to D
2.     C[1,y]←V[1]
3. for k←2 to n
4.     for y←1 to D
5.         C[k,y]←C[k-1,y]
6.         i[k,y]←i[k-1,y]
```

```
7.        if y≥L[k] and C[k-1,y-L[k]]+V[k]>C[k-1,y]
8.        then C[k,y]←C[k-1,y-L[k]]+V[k]
9.             i[k,y]←k
```

算法在第 1 行时间为 $O(D)$，第 3 行和第 4 行的循环进行 $O(nD)$ 次，循环内部是常数时间的操作，于是算法最坏情况下的时间复杂度是 $O(nD)$.

3.4 设任务集 $J=\{1,2,\cdots,n\}$，机器为 M_1 和 M_2. 一个调度是函数 $f: J\to\{1,2\}$，如果 $f(i)=1$，那么任务 i 将分配到 M_1 上；如果 $f(i)=2$，则分配在 M_2 上. 因为任务之间的安排没有偏序约束，所以只要 f 给定，即安排在 M_1 和 M_2 上的任务确定，无论 M_1 和 M_2 上的任务怎样排序，每台机器的加工时间都是不变的. 两台机器的加工时间分别为

$$D_1(f)=\sum_{f(i)=1} t_i,\quad D_2(f)=\sum_{f(i)=2} t_i$$

调度 f 的总加工时间是

$$D(f)=\max\{D_1,D_2\}$$

问题的解是求一个使得 $D(f)$ 达到最小值的调度 f.

命题 3.1 令 $T=\left\lfloor \frac{1}{2}\sum_{i=1}^{n} t_i \right\rfloor$，对于给定输入，一定存在一个最优解 f^*，使得 $D_1(f^*)\leqslant T$ 且 $D_1(f^*)$ 达到最大.

证 假设一个最优解 f 满足 $D_1(f)>D_2(f)$. 交换 M_1 和 M_2 的任务，得到解 f^*，那么 f^* 的总加工时间与 f 的总加工时间相等，因此 f^* 也是最优解且 $D_1(f^*)\leqslant D_2(f^*)$. 由于

$$D_1(f^*)+D_2(f^*)=\sum_{i=1}^{n} t_i$$

且 $D_1(f^*)\leqslant D_2(f^*)$，于是 $D_1(f^*)\leqslant \frac{1}{2}\sum_{i=1}^{n} t_i$. 由于所有任务的加工时间 t_i 都是正整数，因此 $D_1(f^*)\leqslant T$.

f^* 的总加工时间 $D(f^*)=D_2(f^*)$. 作为最优调度，$D_2(f^*)$ 一定达到最小，而 $D_1(f^*)+D_2(f^*)$ 对所有的调度都是一样的，当 $D_2(f^*)$ 达到最小时，$D_1(f^*)$ 必然达到最大.

根据命题 3.1，可以设计一个动态规划算法找一个解 f^*，使得 $D_1(f^*)\leqslant T$ 且 $D_1(f^*)$ 达到最大. 令 $x_i=1$ 当且仅当 $f(i)=1$，原问题变成如下组合优化问题：

$$\max \sum_{i=1}^{n} x_i t_i$$

$$x_i=0,1,\quad i=1,2,\cdots,n$$

$$\sum_{i=1}^{n} t_i x_i \leqslant T,\quad T=\left\lfloor \frac{1}{2}\sum_{i=1}^{n} t_i \right\rfloor$$

这是一个 0-1 背包问题，令 $F[k,y]$ 表示考虑前 k 个任务，在 M_1 的加工时间不超过 y 的情况下其加工时间 $\sum_{i=1}^{k} t_i x_i$ 的最大值. 那么有如下公式：

$$F[k,y]=\max\{F[k-1,y],F[k-1,y-t_k]+t_k\},\quad k>1,\quad T\geqslant y>0$$

$$F[1,y]=\begin{cases} t_1 & \text{如果 } y\geqslant t_1 \\ 0 & \text{否则} \end{cases}$$

$$F[k,0]=0$$
$$F[k,y]=-\infty \quad 如果\ y<0$$

在 $F[k,y]$计算时,设立标记函数 $i[k,y]$

$$i[k,y]=\begin{cases} i[k-1,y] & 如果\ F[k-1,y-t_k]+t_k<F[k-1,y] \\ k & 否则 \end{cases},\ k>1,\ T\geqslant y>0$$

$$i[1,y]=\begin{cases} 1 & y\geqslant t_1 \\ 0 & 否则 \end{cases}$$

最坏情况下的时间复杂度为 $O(nT)$,其中 $T=\left\lfloor \frac{1}{2}\sum_{i=1}^{n} t_i \right\rfloor$.

具体实例的计算过程:

$$T=\left\lfloor \frac{1}{2}(1+5+2+10+3) \right\rfloor=10.5=10$$

备忘录和标记函数如表 3.3 和表 3.4 所示.

表 3.3　$F[k,y]$

k \ y	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	5	6	6	6	6	6
3	1	2	3	3	5	6	7	8	8	8
4	1	2	3	3	5	6	7	8	8	10
5	1	2	3	3	5	6	7	8	9	10

表 3.4　$i[k,y]$

k \ y	1	2	3	4	5	6	7	8	9	10
1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	1	2	2	2	2	2	2
3	1	3	3	3	2	2	3	3	3	3
4	1	3	3	3	2	2	3	3	3	4
5	1	3	3	3	5	5	3	5	5	5

由 $F[5,10]=10$ 可知,M_1 的加工时间 $D_1(f)=10$,于是 f 的总加工时间

$$D_2(f)=21-D_1(f)=21-10=11$$

由 $i[5,10]=5$ 可知 $x_5=1$,接着查

$$i[4,10-t_5]=i[4,7]=3$$

可知 $x_3=1$,接着查

$$i[2,7-t_3]=i[2,5]=2$$

可知 $x_2=1$，接着查

$$i[1,5-t_2]=i[1,0]=0$$

追踪完成，于是调度是

$$f(2)=f(3)=f(5)=1,\quad f(1)=f(4)=2,\quad D_2=11$$

如果 $i[k,y]$ 的定义不同，可能得到另外一个解：

$$f(4)=1,\quad f(1)=f(2)=f(3)=f(5)=2,\quad D_2=11$$

3.5 设 x_i 表示第 i 种硬币使用的个数，$i=1,2,\cdots,n$. 该问题的描述为

$$\min \sum_{i=1}^{n} w_i x_i$$

$$\sum_{i=1}^{n} v_i x_i = y \quad x_i \text{ 为非负整数}$$

令 $F_k(x)$ 表示只允许使用前 k 种钱，总付款为 x 时所使用零钱的最轻重量，则

$$F_k(x)=\min\{F_{k-1}(x),F_k(x-v_k)+w_k\},\quad k>1,\ 0<x\leqslant y$$

$$F_1(x)=w_1\left\lfloor \frac{x}{v_1}\right\rfloor = w_1 x$$

$$F_k(0)=0$$

$$F_k(x)=+\infty,\quad x<0$$

设立标记函数 $t_k(y)$ 记录 $F_k(y)$ 取得最小值时最大币值的标号是否为 k.

$$t_k(x)=\begin{cases} k & F_k(x-v_k)+w_k\leqslant F_{k-1}(x) \\ t_{k-1}(x) & \text{否则} \end{cases},\quad k>1,\ 0<x\leqslant y$$

$$t_1(x)=1,\quad 0<x\leqslant y$$

$$t_k(0)=0,\quad k=1,2,\cdots,n$$

算法的伪码是：

```
Coin
输入：w[1..n],v[1..n],y      //w,v 分别为硬币的重量和币值数组,y 是付款数
输出：F[i,j],t[i,j] i=1,2,…,n,j=1,2,…,y
1. for j←1 to y do
2.     F[1,j]←j * w[1]
3.     t[1,j]←1
4. for i←2 to n do
5.     for  j←1 to y do
6.         F[i,j]←F[i−1,j]
7.         t[i,j]←t[i−1,j]
8.         if F[i,j−v[i]]+w[i]≤F[i−1,j]
9.         then F[i,j]←F[i,j−v[i]]+w[i]
10.             t[i,j]←i
11. return 二维数组 F,t
```

算法的时间复杂度主要取决于第 4 行和第 5 行的 for 循环，内部工作量是常数时间，算法的时间是 $O(ny)$. 通过数组 t 追踪解的过程比较简单，时间不超过 $O(ny)$. 于是算法最坏情况下的时间复杂度是 $O(ny)$.

针对给定实例,算法的备忘录如表 3.5 和表 3.6 所示.

表 3.5 $F_k(x)$

k \ x	1	2	3	4	5	6	7	8	9	10	11	12
1	1	2	3	4	5	6	7	8	9	10	11	12
2	1	2	3	2	3	4	5	4	5	6	7	6
3	1	2	3	2	3	4	5	4	5	6	7	6
4	1	2	3	2	3	4	5	4	5	6	7	6

表 3.6 $t_k(x)$

k \ x	1	2	3	4	5	6	7	8	9	10	11	12
1	1	1	1	1	1	1	1	1	1	1	1	1
2	1	1	1	2	2	2	2	2	2	2	2	2
3	1	1	1	2	2	3	3	2	2	3	3	2
4	1	1	1	2	2	3	3	2	2	3	3	2

问题实例的解是:最轻重量是 6,由 $t_4(12)=2$ 知道 $x_4=0,x_3=0,x_2\geqslant 1$,再由

$$t_2(12-4)=t_2(8)=2\Rightarrow x_2\geqslant 2$$

$$t_2(8-4)=t_2(4)=2\Rightarrow x_2\geqslant 3$$

$$t_2(4-4)=t_2(0)=0\Rightarrow x_2=3,\quad x_1=0$$

从而得到 $x_1=x_3=x_4=0,x_2=3$,只用了 3 枚币值为 4 的硬币.

3.6 使用动态规划算法. 令 $F_k(y)=0,1$,

$F_k(y)=1$ 用前 k 种钱币、总钱数为 y 时可以付款

那么 $F_k(y)$满足如下递推方程:

$$F_k(y)=\max\{F_{k-1}(y),F_{k-1}(y-v_k)\},\quad k=2,3,\cdots,n,\ y=1,2,\cdots,M$$

$$F_1(y)=\begin{cases}1 & 若\ y=v_1\\ 0 & 否则\end{cases}$$

$$F_k(y)=0\quad 若\ y<0,\ k=2,3,\cdots,M$$

设立标记函数 $i_k(y)$

$$i_k(y)=\begin{cases}k & 若\ F_{k-1}(y-v_k)=1\\ i_{k-1}(y) & 否则\end{cases}$$

算法计算出 $F_n(M)$. 如果 $F_n(M)=1$,则存在一种找零钱的方法. 所使用的零钱币值可根据 $i_k(n)$的值向前追踪得到. 算法的时间复杂度为 $T(n)=O(nM)$.

3.7 (1) 设 $y_i=1$ 表示选择了位置 i,$y_i=0$ 表示没有选位置 i,问题的目标函数和约束条件是:

$$\max\sum_{i=1}^{n}p_iy_i$$

$$|x_i-x_j|\geqslant d,\quad i,j=1,2,\cdots,n,\ i\neq j,\ y_i=y_j=1$$

(2) 排序使得 $x_1<x_2<\cdots<x_n$，$F_k(y)$表示考虑前 k 个位置，距原点最远到 y 的范围内开设商店的最大收益. 那么 $F_k(y)$满足如下递推方程：

$$F_k(y)=\begin{cases}\max\{F_{k-1}(y),F_{k-1}(x_k-d)+p_k\} & \text{当 } x_k\leqslant y\\ F_{k-1}(y) & \text{否则}\end{cases}$$

$$F_1(y)=\begin{cases}p_1 & \text{当 } x_1\leqslant y\\ 0 & \text{否则}\end{cases}$$

如下定义标记函数：

$$i_k(y)=\begin{cases}k & \text{当 } F_{k-1}(x_k-d)+p_k\geqslant F_{k-1}(y)\\ i_{k-1}(y) & \text{否则}\end{cases}$$

$$i_1(y)=\begin{cases}1 & \text{当 } x_1\leqslant y\\ 0 & \text{否则}\end{cases}$$

算法的时间复杂度为 $O(nD+n\log n)$，其中 $D=\max\{x_i\mid i=1,2,\cdots,n\}$.

3.8 令 $B=N/2$，那么 $\min\{\sum_{a_i\in A_1}a_i,\sum_{a_j\in A_2}a_j\}\leqslant B$. 不妨设 $\sum_{a_i\in A_1}a_i\leqslant\sum_{a_j\in A_2}a_j$. 那么问题变成求 A 的子集 A_1 使得

$$\max\sum_{a_i\in A_1}a_i$$

$$\sum_{a_i\in A_1}a_i\leqslant B$$

那么该问题等价于双机调度问题(习题 3.4). 通过动态规划算法求出最优解 A_1，如果 A_1 中的数之和 $\sum_{a_i\in A_1}a_i=B$，那么输出“Yes”，否则输出“No”. 该算法的时间复杂度为 $O(nN)$.

3.9 与 0-1 背包问题类似，使用动态规划方法. 令 $N_j(d)$表示考虑作业集$\{1,2,\cdots,j\}$、结束时间为 d 的最优调度的效益，那么

$$N_j(d)=\begin{cases}\max\{N_{j-1}(d),N_{j-1}(d-t(j))+v_j\} & d\geqslant t(j)\\ N_{j-1}(d) & d<t(j)\end{cases}$$

$$N_1(d)=\begin{cases}v_1 & t(1)\leqslant d\\ 0 & t(1)>d\end{cases}$$

$$N_j(0)=0$$

$$N_j(d)=-\infty \quad d<0$$

自底向上计算，存储使用备忘录方法. 可以使用标记函数 $B(j)$记录使得 $N_j(d)$达到最大时是否

$$N_{j-1}(d-t(j))+v_j>N_{j-1}(d)$$

如果是，则 $B(j)=j$；否则 $B(j)=B(j-1)$.

算法的伪码是

输入：加工时间 $t[1..n]$，效益 $v[1..n]$，结束时间 D

输出：最优效益 $N[i,j]$，标记函数 $B[i,j]$，$i=1,2,\cdots n$，$j=1,2,\cdots,D$

```
1.  for d←1 to t[1]−1
2.      N[1,d]←0, B[1]←0
3.  for d←t[1] to D
```

```
4.        N[1,d]←v[1],B[1]←1
5.   for j←2 to n
6.        for d←1 to D
7.             N[j,d]←N[j-1,d]
8.             B[j,d]←B[j-1,d]
9.             if d≥t[j] and N[j-1,d-t[j]]+v[j]>N[j-1,d]
10.             then  N[j,d]←N[j-1,d-t[j]]+v[j]
11.                   B[j,d]←j
```

得到最大效益 $N[n,D]$后,通过对 $B[n,D]$的追踪,可以得到问题的解. 算法的主要工作是第 5 行和第 6 行的 for 循环,需要执行 $O(nD)$次,循环体内的工作量是常数,追踪解的工作量不大,于是算法最坏情况下的时间复杂度是 $O(nD)$.

3.10 设 x_i 表示装入背包的第 i 种物品个数,$i=1,2,\cdots,n$,推广的 0-1 背包问题的描述是

$$\max \sum_{i=1}^{n} v_i x_i$$

$$\sum_{i=1}^{n} w_i x_i \leqslant W$$

$$\sum_{i=1}^{n} c_i x_i \leqslant V$$

$$x_i = 0,1$$

设 $m[i,j,k]$表示使用前 i 种物品、背包重量限制为 j、容积为 k 时的最大价值,其中 $i=1,2,\cdots,n$,$j=1,2,\cdots,W$,$k=1,2,\cdots,V$,那么递推方程是:

$$m[i,j,k]=\begin{cases}\max\{m[i-1,j,k],m[i-1,j-w_i,k-c_i]+v_i\} & 若\ j\geqslant w_i\ 且\ k\geqslant c_i\\ m[i-1,j,k] & 否则\end{cases}$$

$$i=2,\cdots,n,\quad j=1,2,\cdots,W,\quad k=1,2,\cdots,V$$

$$m[1,j,k]=\begin{cases}v_1 & 如果\ j\geqslant w_1\ 且\ k\geqslant c_1\\ 0 & 否则\end{cases}$$

$$m[i,0,k]=m[i,j,0]=0$$

$$m[i,j,k]=-\infty \quad j<0\ 或\ k<0$$

与前面的背包问题类似,可定义标记函数 $t[i,j,k]$用于追踪问题的解. 其中

$$t[i,j,k]=\begin{cases}i & 如果\ m[i-1,j,k]\leqslant m[i-1,j-w_i,k-c_i]+v_i,j\geqslant w_i,k\geqslant c_i\\ t[i-1,j,k] & 否则\end{cases}$$

$$i=2,\cdots,n,\quad j=1,2,\cdots,W,\quad k=1,2,\cdots,V$$

$$t[1,j,k]=\begin{cases}1 & 如果\ j\geqslant w_1\ 且\ k\geqslant c_1\\ 0 & 否则\end{cases}$$

该算法最坏情况下的时间复杂度是 $O(nWV)$.

3.11 设 $X=\{x_0,x_1,\cdots,x_{n-1}\}$顺时针排列在圆环上. 类似于矩阵链乘法的动态规划算法,用 i 和 j 界定子问题的边界. X_{ij} 表示按顺时针合并$\{x_i,\cdots,x_j\}$后的数组,其含有的整数个数是 n_{ij},完成合并在最坏情况下所需的最少比较次数记作 $m[i,j]$,其中 $i=0,1,\cdots,n-1$,$j=0,1,\cdots,n-1$.

考虑数组 X_{ik} 和 $X_{(k+1)j}$ 合并成 X_{ij} 的比较次数. 因为它们都是排好序的,比较次数至多是元素总数减 1,于是合并这两个数组的比较次数在最坏情况下是

$$n_{ik} + n_{(k+1)j} - 1$$

与矩阵链乘法的不同在于:矩阵链的子问题的边界是排列成一条线,保持 $i \leqslant j$;而这里是环,可能出现 $i > j$ 的情况. 如图 3.3 所示,左边的合并对应的是 $i < j$ 的子问题,右边的合并对应的是 $i > j$ 的子问题. 在 $i > j$ 时相应的递推方程就不一样了.

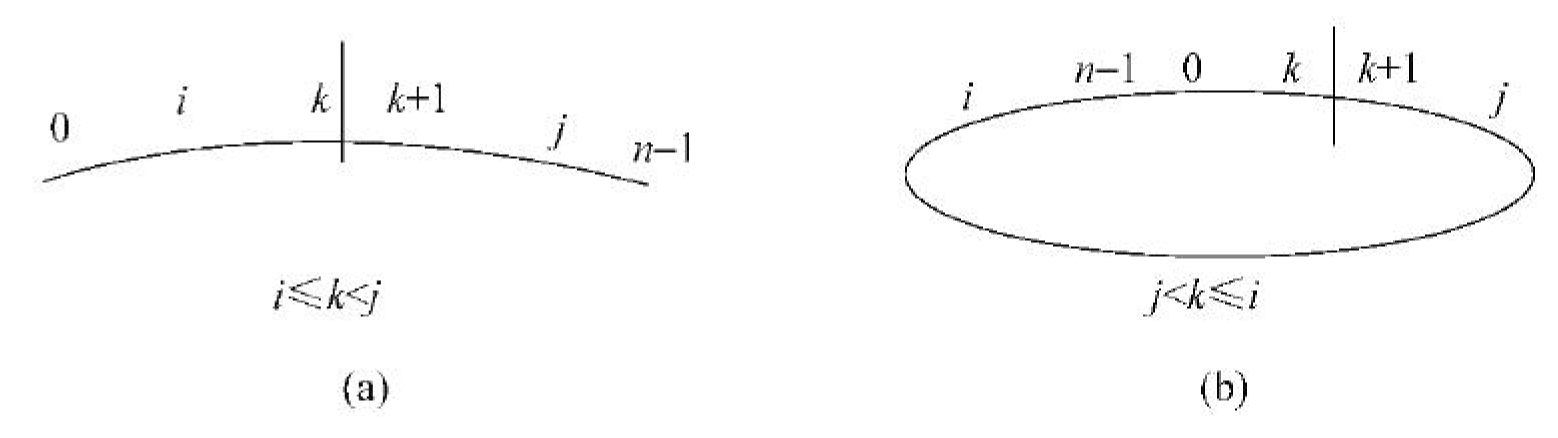


图 3.3 子问题的归约

令 $m[i,j]$ 表示在最坏情况下合并成 X_{ij} 所需要的最少的比较次数. 考虑在 $i < j$ 的情况,X_{ik} 与 $X_{(k+1)j}$ 合并的比较次数至多是 $n_{ik} + n_{(k+1)j} - 1$,于是有

$$m[i,j] = \min_{i \leqslant k < j}\{m[i,k] + m[k+1,j]\} + \sum_{t=i}^{j} n_t - 1$$

当 $i > j$ 时,如图 3.3(b)所示,当 k 取 $n-1$ 时,$k+1=0$,这时候优化函数的和应该是

$$m[i,n-1] + m[0,j] = m[i,k] + m[(k+1)\bmod n, j]$$

而为计算合并的比较次数,需要分成两段求和,即

$$\sum_{t=i}^{n-1} n_t + \sum_{t=0}^{j} n_t - 1$$

于是有

$$m[i,j] = \min_{i \leqslant k < j}\{m[i,k] + m[k+1,j]\} + \sum_{t=i}^{j} n_t - 1, \quad i < j$$

$$m[i,j] = \min_{\substack{i \leqslant k \leqslant n-1 \\ 0 \leqslant k < j}}\{m[i,k] + m[(k+1)\bmod n, j]\} + \sum_{t=i}^{n-1} n_t + \sum_{t=0}^{j} n_t - 1, \quad i > j$$

$$m[i,i] = 0, \quad i = 0,1,\cdots,n-1$$

上述公式中的 $m[i,i]$ 是递推关系的初值. 算法将从规模为 1 的子问题开始迭代计算,直到规模为 n 的问题为止. 在矩阵链相乘问题中,规模为 n 的子问题只有一个;而在环形排列的合并问题中,规模为 n 的子问题可能以 $i=0,1,\cdots,n-1$ 中的任何位置为起点,恰好有 n 个. 我们需要从这 n 个子问题的解中找到具有最小比较次数的解. 于是最少的比较次数是

$$m = \min_{0 \leqslant i \leqslant n-1}\{m[i,(i+n-1)\bmod n]\}$$

与矩阵链乘法相似,计算 $m[i,j]$ 过程中需要用标记函数 $s[i,j]$ 记录使得 $m[i,j]$ 取得最小值的 k. 以便找到最优的合并次序. 与矩阵链链乘法问题类似,该算法最坏情况下的时间复杂度是 $O(n^3)$.

3.12 如图 3.4 所示,n 边形的顶点是 $1,2,\cdots,n$. 顶点 $i-1,i,\cdots,j$ 构成的凸多边形记作 $A[i,j]$,于是原始问题就是 $A[2,n]$.

考虑子问题 $A[i,j]$ 的划分,假设它的所有划分方案中的最小权值是 $t[i,j]$. 从 $i,i+1,\cdots,j-1$ 中任选顶点 k,它与底边 $(i-1)j$ 构成一个三角形(图 3.4 中的灰色三角形). 这个三角形将 $A[i,j]$ 划分成两个凸多边形: $A[i,k]$ 和 $A[k+1,j]$,从而产生了两个子问题. 这两个凸多边形的划分方案的最小权值分别是 $t[i,k]$ 和 $t[k+1,j]$. 根据动态规划思想,$A[i,j]$ 相对于这个 k 的划分方案的最小权值是

$$t[i,k]+t[k+1,j]+d_{(i-1)k}+d_{kj}+d_{(i-1)j}$$

其中 $d_{(i-1)k}+d_{kj}+d_{(i-1)j}$ 是三角形 $(i-1)kj$ 的周长,于是得到递推关系:

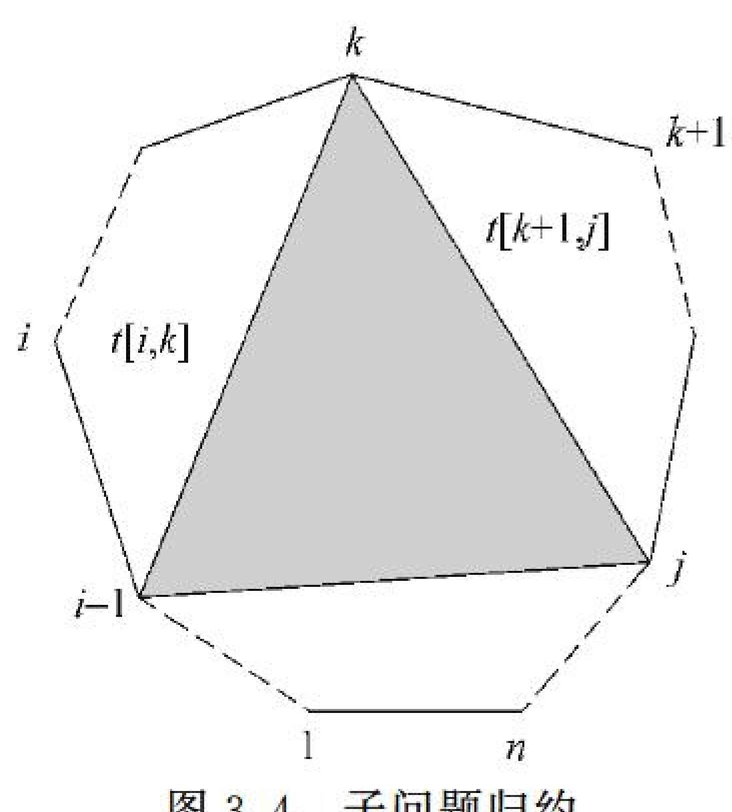


图 3.4 子问题归约

$$t[i,j]=\begin{cases}0 & i=j \\ \min\limits_{i\leqslant k<j}\{t[i,k]+t[k+1,j]+d_{(i-1)k}+d_{kj}+d_{(i-1)j}\} & i<j\end{cases}$$

$$t[i,i]=0$$

不难看出,这个递推关系与矩阵链相乘算法的递推式非常相似,可以定义标记函数记下得到最小权值的 k 的位置,算法最坏情况下的时间复杂度是 $O(n^3)$.

3.13 子问题分别对应于 $M_0,M_1,\cdots,M_n$ 的计算. M_0 是输入. 在第 k 步计算 M_{k+1} 的关键是找到它与前面矩阵的依赖关系. 假设 M_k 已经计算完毕,如何计算 M_{k+1} 呢? 这需要对于每组 i,j,确定 $M_{k+1}[i,j]$ 是否为 1. 条件是:

$M_{k+1}[i,j]=1\Leftrightarrow$ 在 D 中存在一条从 x_i 到 x_j 且只经过 $\{x_1,x_2,\cdots,x_k,x_{k+1}\}$ 中顶点的路径. 可以将这种路径分成两类:

第一类是只经过 $\{x_1,x_2,\cdots,x_k\}$ 中顶点的路径,这时 $M_k[i,j]=1$.

第二类是经过顶点 x_{k+1} 的路径. 因为回路可以从路径中删除,因此只需考虑经过 x_{k+1} 一次的路径. 这条路径可以分成两段,从 x_i 到 x_{k+1},再从 x_{k+1} 到 x_j,因此有 $M_k[i,k+1]=1$ 和 $M_k[k+1,j]=1$. 这就是对于第二类路径的判别条件,即:

$$M_{k+1}[i,j]=1\Leftrightarrow M_k[i,k+1]=1 \wedge M_k[k+1,j]=1$$

算法 Warshall

输入: M //图 D 的邻接矩阵

输出: M_t //图 D 的连通矩阵

```
1.  Mt ← M
2.  for k ← 1 to n do
3.      for i ← 1 to n do
4.          for j ← 1 to n do
5.              Mt[i,j] ← Mt[i,j] + Mt[i,k] * Mt[k,j]
```

注意,上述算法中矩阵加法和乘法 * 中的元素相加都使用逻辑加,即 1+0=0+1=1+1=1,0+0=0.

下面分析算法的时间复杂度,在 Warshall 算法中,第 2 行、第 3 行、第 4 行都是 n 步的循环,因此第 5 行总共被执行 n^3 次,而每次只做 1 次乘法和 1 次加法,因此 Warshall 算法的

时间复杂度是 $O(n^3)$.

对于给定实例,利用 Warshall 算法计算的矩阵序列如下所示:

$$M_0=\begin{bmatrix}0&1&1&0\\0&0&1&0\\0&0&0&1\\0&0&1&0\end{bmatrix},\quad M_1=\begin{bmatrix}0&1&1&0\\0&0&1&0\\0&0&0&1\\0&0&1&0\end{bmatrix},\quad M_2=\begin{bmatrix}0&1&1&0\\0&0&1&0\\0&0&0&1\\0&0&1&0\end{bmatrix}$$

$$M_3=\begin{bmatrix}0&1&1&1\\0&0&1&1\\0&0&0&1\\0&0&1&1\end{bmatrix},\quad M_4=\begin{bmatrix}0&1&1&1\\0&0&1&1\\0&0&1&1\\0&0&1&1\end{bmatrix}$$

因此,a 可以连通到 b、c 和 d;b 可以连通到 c 和 d;c 可以连通到 d; d 可以连通到 c.

3.14 使用动态规划算法求解这个问题. 先考虑子问题划分. 设 $S_1[1..i]$和 $S_2[1..j]$表示两个子序列,$C[i,j]$表示 $S_1[1..i]$和 $S_2[1..j]$的最小编辑距离. 考虑最后一对字符,如果 $S_1[i]$被删除,那么最小编辑距离是 $S_1[i-1]$和 $S_2[j]$的最小编辑距离加 d;如果在 $S_1[i]$后面插入 $S_2[j]$,那么编辑距离是 $S_1[i]$和 $S_2[j-1]$的最小编辑距离加 d;如果 $S_1[i]$被替换成 $S_2[j]$,那么最小编辑距离是 $S_1[i-1]$与 $S_2[j-1]$的最小编辑距离加 r;如果 $S_1[i]=S_2[j]$,那么最小编辑距离是 $S_1[i-1]$与 $S_2[j-1]$的最小编辑距离. 递推关系如下:

$$C[i,j]=\min\{C[i-1,j]+d,C[i,j-1]+d,C[i-1,j-1]+t[i,j]\}$$
$$i=1,2,\cdots,n,\quad j=1,2,\cdots,m$$
$$t[i,j]=\begin{cases}0, & S_1[i]=S_2[j]\\ r, & S_1[i]\neq S_2[j]\end{cases}$$
$$C[0,j]=jd$$
$$C[i,0]=id$$

可以定义标记函数记录得到最优 $C[i,j]$时所对应的选择(删除、插入、替换等),算法的时间复杂度是 $O(nm)$.

3.15 **方法一**:设 $F[j]$表示第 $1,2,\cdots,j$ 天加工任务的最大数目. 如果在第 i 天进行检修,那么从第 $i+1$ 天到第 j 天连续工作的加工量用 $w[i+1,j]$表示,则

$$w[i+1,j]=\sum_{k=i+1}^{j}\min\{x_k,s_{k-i}\},\quad 0\leqslant i<j\leqslant n$$

不难看出,$w[i+1,j]$满足如下递推方程:

$$w[i+1,j]=w[i+1,j-1]+\min\{x_j,s_{j-i}\},\quad 0\leqslant i<j\leqslant n,\ j>1$$
$$w[i+1,i+1]=\min\{x_{i+1},s_1\},\quad i=0,1,\cdots,n$$

假设最后一次检修机器在第 i 天,那么从第 $i+1$ 天到第 j 天连续加工任务总数为 $w[i+1,j]$,而检修前加工任务数至多是 $F[i-1]$. 于是当最后一次检修发生在第 i 天时,到第 j 天为止,加工任务数至多是 $F[i-1]+w[i+1,j]$. 考虑到所有可能的检修时间 $i(i=1,2,\cdots,j-1)$,或者在第 j 天之前根本不做检修的情况,可以得到如下递推关系:

$$F[j]=\max\begin{cases}\max\limits_{0<i<j}\{F[i-1]+w[i+1,j]\} & j>1\\ w[1,j] & j>0\end{cases}$$
$$F[0]=0$$

可以定义标记函数记录得到最优 $F[j]$时的检修时间 i. 如果每次都重新计算 $w[i+1,j]$,算法最坏情况下的时间复杂度是 $O(n^3)$. 如果在预处理时根据 $w[i.j]$的递推方程计算所有的 $w[i,j]$,$1\leqslant i\leqslant j\leqslant n$,并把计算结果存成备忘录,在计算 $F[j]$时直接查找相应的值,那么计算 $F[j]$的时间可以降为 $O(n^2)$,而预处理的时间也是 $O(n^2)$,因此算法最坏情况下的时间复杂度是 $O(n^2)$.

方法二:参照矩阵链相乘的方式划分子问题,该算法的时间复杂度高一些.

首先证明下述命题.

命题 3.2 在最优解中不会出现连续两天检修的情况.

证 假若不然,在一个最优解 T 的第 i 和 $i+1$ 天都进行检修. 那么,当去掉第 i 天的检修后,不会减少从第 $i+1$ 天到第 n 天的加工量;也不会减少第 1 天到第 $i-1$ 天的加工量;只有第 i 天的加工量由 0 增加到 $\min\{x_i,s_t\}$,其中 s_t 表示第 i 天机器具有的加工能力. 于是总加工量将大于原来的加工量,从而与 T 的最优性矛盾.

通过类似的分析还可以证明,检修也不会发生在子问题的最后一天.

利用上述性质,可以从检修的时间点 k 将原问题划分成两个子问题. 设 $G[i,j]$表示第 i 天到第 j 天的最大加工任务数. 如果在第 k 天进行检修,$i<k<j$,那么最大加工任务数等于 $G[i,k-1]+G[k+1,j]$. 如果在第 i 天到第 j 天的时间内没有发生检修,那么加工任务数是 $w[i,j]$,于是得到如下递推方程:

$$G[i,j]=\max\{w[i,j],\max_{i<k<j}\{G[i,k-1]+G[k+1,j]\}, \quad 1\leqslant i<k<j\leqslant n$$

$$G[i,i]=w[i,i], \quad i=1,2,\cdots,n$$

标记函数的设定可参照矩阵链相乘问题的做法,只是当最优解的第 i 到第 j 天之间没有进行检修时将标记函数设为 0. 该算法最坏情况下的时间复杂度是 $O(n^3)$.

3.16 设 $x_1,x_2,\cdots,x_n$ 是 0 或 1,$x_i=1$ 当且仅当下载请求 i 被批准. 那么

$$\max \sum_{i=1}^{n} a_i x_i$$

$$\sum_{i=1}^{n} a_i x_i \leqslant K, \quad x_i=0,1$$

类似于 0-1 背包问题,令 $F_i(y)$表示考虑前 i 个申请,带宽限制为 y 时的最大带宽使用量,则有如下递推式:

$$F_i(y)=\max\{F_{i-1}(y),F_{i-1}(y-a_i)+a_i\}, \quad i>1,\ 0<y\leqslant K$$

$$F_1(y)=\begin{cases}a_1, & y\geqslant a_1\\ 0, & y<a_1\end{cases}$$

$$F_i(0)=0$$

$$F_i(y)=-\infty, \quad y<0$$

与前面背包问题类似可设立标记函数,自底向上顺序处理 $i=1,2,\cdots,n$ 的情况. 对于给定的 i,令 $y=1,2,\cdots,K$,依次确定各个子问题的 $F_i(y)$值,最后由标记函数得到 x_i 的值. 算法时间复杂度为 $O(nK)$.

3.17 令 $F[i,j]$表示 a_{11} 到 a_{ij} 的路径上的数的最小和,则

$$F[i,j]=\min\{F[i-1,j],F[i-1,j-1]\}+a_{ij}, \quad i=2,3,\cdots,n,\ j=2,3,\cdots,i-1$$

$$F[i,1]=F[i-1,1]+a_{i1},\quad i=2,3,\cdots,n$$
$$F[i,i]=F[i-1,i-1]+a_{ii},\quad i=2,3,\cdots,n$$
$$F[1,1]=a_{11}$$

问题的最优路径上的和是

$$\min\{F[n,j]\mid j=1,2,\cdots,n\}$$

可以定义标记函数 $k[i,j]$，记录得到最优 $F[i,j]$时的路径选择，即

$$k[i,j]=\begin{cases}j & \text{若 } F[i-1,j]<F[i-1,j-1]\\ j-1 & \text{否则}\end{cases}$$

算法的时间复杂度为 $O(n^2)$.

3.18 本题不是优化问题. 也可以使用动态规划的设计思想，通过多阶段决策来求解. 与一般优化问题的区别在于：当把一个问题的计算与子问题的计算建立递推关系的时候，不是只考虑达到最优的那个子问题的结果，而是考虑所有可能的子问题的结果. 考虑到每个串计算的结果至多是 3 种，即 a、b 或 c，为了后面的计算，需要把它们全部保留下来. 因此，可以把记录优化函数的一个极大或极小值推广到记录有限个值. 只要用含有 3 个项的数组替换备忘录中的一个项，就可以使用类似于矩阵链相乘的方法划分子问题，并利用动态规划方法设计算法.

令 $X[i,j]$是串 $x_i\cdots x_j$ 的可能的运算结果(由于计算顺序不同，结果可能不唯一，但至多为 3 个结果，即 a,b,c). 设计存储 $X[i,j]$的结构为 3×2 阵列，分别记录 3 个可能的运算结果(可能取 a、b 或 c)和得到这个结果时所对应的划分位置 k. 自底向上计算，通过较小子问题的结果来计算较大子问题的结果，其递推公式是：

$$X[i,j]=X[i,k]X[k+1.j],\quad 1\leqslant i<j\leqslant n,\ k=i,i+1,\cdots,j-1$$
$$X[i,i]=x_i,\quad i=1,2,\cdots,n$$

伪码如下：

```
for i←1 to n
    X[i,i]←x_i
for r←2 to n−1                          //计算所有大小为 r 的子问题
    for i←1  to n−r+1                   //子问题的前边界 i
        j←i+r−1                         //子问题的后边界 j
        for k←i to j−1                  //在 k 位置划分成两个更小的子问题
            X[i,j]←X[i,k]X[k+1,j]
                              //记录 X[i,j]可能的结果和划分位置 k，至多为 3 个
for k←1 to n−1                          //规模为 n 的原始问题
    if X[1,k]X[k+1,n]="a" then return "1"   //存在计算顺序结果为 a
return "0"                              //没有结果等于 a
```

与矩阵链计算类似，可通过所记录的 k 值追踪出相应的运算顺序. 该算法最坏情况下的时间复杂度为 $O(n^3)$.

第 4 章 贪心法

4.1 内容提要

1. 基本概念

贪心法(Greedy Approach)是一种求解组合优化问题的算法设计技术,其求解过程由一系列决策构成,每一步决策仅依赖于某种局部优化的性质. 与动态规划算法不同,贪心法在做决策时不必考虑所有子问题的选择结果.

贪心法的适用条件 问题的求解可以由一系列的决策步骤构成,每步决策依赖于某种局部最优的贪心策略. 正确的贪心策略要保证每一步基于局部优化性质的选择最终导致全局的最优解. 如果不具有上述性质,贪心法对某些实例只能得到近似解.

主要设计步骤

(1) 将问题的求解看作一系列的决策过程.

(2) 确定每一步决策所依据的局部优化性质.

(3) 证明每一步基于局部优化性质的选择最终导致全局最优解.

2. 某些常用的贪心法的正确性证明方法

方法一:数学归纳法,主要步骤如下:

(1) 叙述一个论证算法正确性的与自然数相关的命题 $P(n)$,这里的 n 可代表算法步数或实例规模. 例如:

对于任何正整数 n,贪心法的前 n 步选择将导致最优解.

对于任何正整数 n,贪心法对于规模为 n 的任何实例都得到最优解.

(2) 使用第一或第二数学归纳法证明上述命题 $P(n)$.

第一数学归纳法

证:$P(1)$为真;

若 $P(n)$为真,则 $P(n+1)$为真.

第二数学归纳法

证:$P(1)$为真;

若对所有 $k<n$,有 $P(k)$为真,则 $P(n)$为真.

方法二:交换论证. 主要步骤是:

(1) 分析一般最优解与贪心法的解的区别,然后定义一种转换规则,使得从任意一个最优解出发,经过不断对解的某些成分的排列次序进行交换或者用其他元素替换,将这个解最

终能够转变成贪心法的解.

(2) 证明在上述转换中解的优化函数值不会变坏.

(3) 证明上述转换在有限步结束.

3. 典型的贪心法

活动选择 设 $S=\{1,2,\cdots,n\}$ 为活动的集合，s_i 和 f_i 分别为活动 i 的开始和截止时间，$i=1,2,\cdots,n$. 定义

$$\text{活动 } i \text{ 与 } j \text{ 相容} \Leftrightarrow s_i \geqslant f_j \text{ 或 } s_j \geqslant f_i,\ i \neq j$$

求 S 的最大的两两相容的活动子集 A.

贪心策略：把活动按照截止时间从小到大排序，使得 $f_1 \leqslant f_2 \leqslant \cdots \leqslant f_n$，然后把第 1 项活动加入 A. 接着继续按照标号从小到大挑选，只要与 A 中已选好的活动相容，就可以把这项活动放入 A，直到检查完所有的活动为止.

算法最坏情况下的时间复杂度为 $O(n\log n)$.

装载问题 有 n 个集装箱需要装入最大载重量为 C 的轮船，其中集装箱 i 的重量是 $w_i \leqslant C, i=1,2,\cdots,n$，且无体积限制. 问如何选择而使得装上船的集装箱个数最多?

贪心策略：先对集装箱排序，使得 $w_1 \leqslant w_2 \leqslant \cdots \leqslant w_n$，然后按照标号从小到大顺序装船，直到装入某个箱子 k 使得 $\sum_{i=1}^{k} w_i \leqslant C < \sum_{i=1}^{k+1} w_i$.

算法最坏情况下的时间复杂度为 $O(n\log n)$.

最大延迟调度 给定客户集合 $A=\{1,2,\cdots,n\}$，服务时间集合 $T=<t_1,t_2,\cdots,t_n>$，截止时间集合 $D=<d_1,d_2,\cdots,d_n>$，求使得最大延迟达到最小的调度，即求函数 $f: A \to \mathbf{N}$，使得

$$\min_f\{\max_{i\in A}\{f(i)+t_i-d_i\}\}$$

$$\forall\, i,j \in A, i \neq j, f(i)+t_i \leqslant f(j) \text{ 或 } f(j)+t_j \leqslant f(i)$$

贪心策略：按照截止时间 d_i 从小到大选择任务，在安排时不留空闲时间.

算法最坏情况下的时间复杂度为 $O(n\log n)$.

最优前缀码 给定字符集 $C=\{x_1,x_2,\cdots,x_n\}$ 和每个字符的频率 $f(x_i), i=1,2,\cdots,n$，求关于 C 的最优二元前缀码.

贪心策略：每次选择频率最小的两个字符作为兄弟生成父结点，父结点的频率是这两个结点的频率之和，直到树根为止.

Huffman 算法

输入：$C=\{x_1,x_2,\cdots,x_n\}$ 是字符集，每个字符频率 $f(x_i)$，$i=1,2,\cdots,n$

输出：Q //队列

```
1. n←|C|
2. Q←C                                  //按频率递增构成队列 Q
3. for i←1 to n−1 do
4.     z←Allocate-Node()                //生成结点 z
5.     z.left←Q 中最小元                //取出 Q 中最小元作为 z 的左儿子
6.     z.right←Q 中最小元               //取出 Q 中最小元作为 z 的右儿子
```

```
        f(z)←f(x)+f(y)
        Insert(Q,z)                              //将 z 插入 Q
return Q
```

该算法最坏情况下的时间复杂度是 $O(n\log n)$.

最小生成树 给定连通带权图 $G=\langle V,E,W\rangle$,求 G 的一棵具有最小权的生成树.

(1) Prim 算法.

贪心策略:每次选择连接 S 和 $V-S$ 的权值最小的边 e,将 e 加入树 T,并将 e 关联的顶点 $j\in V-S$ 加入 S,直到 $S=V$ 为止.

Prim 算法

输入:连通图 $G=\langle V,E,W\rangle$

输出:G 的最小生成树 T

```
S←{1};T←∅
while V-S≠∅ do
    从 V-S 中选择 j 使得 j 到 S 中顶点的边 e 的权值最小; T←T∪{e}
    S←S∪{j}
```

算法最坏情况下的时间复杂度是 $O(n^2)$.

(2) Kruskal 算法.

贪心策略:每次选择当前的最短边,只要与树 T 中的边不构成回路,就把它加入,直到 T 中含有 $n-1$ 条边为止.

Kruskal 算法

输入:连通图 G //顶点数 n,边数 m

输出:G 的最小生成树

```
按照权从小到大顺序排序 G 中的边,使得 E={e1,e2,…,em}
T←∅
repeat
    e←E 中的最短边
    if e 的两端点不在同一个连通分支
    then T←T∪{e}
        E←E-{e}
until T 包含了 n-1 条边
```

算法最坏情况下的时间复杂度是 $O(m\log m)$.

单源最短路径 在一个带权有向网络 $G=(V,E,W)$中,每条边 $e=\langle i,j\rangle$的权 $w(e)$为非负实数,表示从 i 到 j 的距离. 网络中有源点 $s\in V$,求从 s 出发到达每个其他结点的最短路径.

贪心策略:每次从 $V-S$ 中选择顶点 j 使得从源点 s 到 j 且中间只经过 S 中的顶点的路径长度最短,该路径连接 j 的顶点是 k. 记录 k,并将 j 加入 S,直到 $S=V$ 为止.

Dijkstra 算法

输入:带权有向图 $G=\langle V,E,W\rangle$,源点 $s\in V$

输出:数组 L,对所有的 $j\in V-\{s\}$,$L[j]$表示 s 到 j 的最短路径上 j 前一个结点的标号

```
S←{s}
dist[s]←0
for i∈V-{s}  do
    dist[i]←w(s,i)                    //如果 s 到 i 没有边,w(s,i)=∞
while V-S≠∅ do
    从 V-S 中取出具有相对 S 的最短路径的顶点 j,k 是该路径上连接 j 的顶点
    S←S∪{j};L[j]←k
      for i∈V-S do
          if dist[j]+w(j,i)<dist[i]
          then dist[i]←dist[j]+w(j,i)     //修改结点 i 相对 S 最短路径长度
```

算法最坏情况下的时间复杂度是 $O(n^2)$.

4.2 习　　题

4.1 设有 n 个顾客同时等待一项服务,顾客 i 需要的服务时间为 $t_i, i=1,2,\cdots,n$. 从时刻 0 开始计时. 若在时刻 t 开始对顾客 i 服务,那么 i 的等待时间就是 t. 应该怎样安排 n 个顾客的服务次序使得总的等待时间(每个顾客等待时间的总和)最少?

(1) 使用贪心法求解这个问题时的贪心选择策略是什么?

(2) 简单写出贪心法的算法描述.

(3) 假设服务时间分别为{1,3,2,15,10,6,12},用贪心法给出这个问题的解.

4.2 有 n 个底面为长方形的物品需要租用库房存放. 如果每个物品都必须放在地面上,且所有物品的底面宽度都等于库房的宽度,那么第 i 个物品占用库房面积大小只需要用它的底面长度 l_i 来表示,$i=1,2,\cdots,n$. 设库房总长度是 L,且 $\sum_{i=1}^{n} l_i > L$. 如果要求放入库房的物品个数最多,选用哪种算法设计技术? 简述算法的设计思想,证明算法的正确性,并估计算法最坏情况下的时间复杂度.

4.3 设有一条边远山区的道路 AB,沿着道路 AB 分布着 n 所房子. 这些房子到 A 的距离分别是 $d_1, d_2, \cdots, d_n$ ($d_1 < d_2 < \cdots < d_n$). 为了给所有房子的用户提供移动电话服务,需要在这条道路上设置一些基站. 为了保证通信质量,每所房子应该位于距离某个基站的 4 千米范围之内. 设计一个算法找到基站的位置,并且使得基站总数达到最少. 用文字说明算法的主要设计思想,给出算法的伪码描述,证明算法的正确性并给出算法最坏情况下的时间复杂度函数.

4.4 给定数轴 X 上 n 个不同点的集合 $\{x_1, x_2, \cdots, x_n\}$,其中 $x_1 < x_2 < \cdots < x_n$. 现在用若干个长度为 1 的闭区间来覆盖这些点. 设计一个算法找到最少的闭区间个数和位置,证明算法的正确性并估计算法的时间复杂度.

4.5 有 n 个文件需要存储在磁盘上,第 i 个文件需要 p_i 个字节的存储空间,$i=1,2,\cdots,n$. 磁盘的总容量是 C,且 $\sum_{i=1}^{n} p_i > C$.

(1) 如果要求存入的文件个数达到最多,选用哪种算法设计技术? 简述算法设计思想,

证明算法的正确性,并估计算法在最坏情况下的时间复杂度.

(2) 如果要求磁盘的剩余空间达到最小,选用哪种算法设计技术? 简述算法设计思想,并估计算法最坏情况下的时间复杂度.

4.6 有 n 项作业的集合 $J=\{1,2,\cdots,n\}$,每项作业 i 有加工时间 $t(i)\in\mathbf{Z}^+$. 有一台机器从时刻 0 开始工作,直到完成所有的任务. 一个可行调度 f 是对 J 中任务的一个安排,对于 $i\in J$,$f(i)$是任务 i 开始加工的时间,f 满足下述条件:

$$f(i)+t(i)\leqslant f(j) \text{ 或 } f(j)+t(j)\leqslant f(i),\ j\neq i,\ i,j\in J$$

设作业 i 的完成时间 $w(i)=f(i)+t(i)$,求使得平均完成时间 $\frac{1}{n}\sum_{i=1}^{n}w(i)$ 最少的调度.

4.7 假设零钱系统的币值是$\{1,p,p^2,\cdots,p^n\}$,$p>1$,且每个钱币的重量都等于 1. 设计一个最坏情况下时间复杂度最低的算法,使得对任何钱数 y,该算法得到的零钱个数最少. 说明算法的主要设计思想,证明它的正确性,并给出最坏情况下的时间复杂度.

4.8 有一个考察队到野外考察,在考察路线上有 n 个地点可以作为宿营地. 已知宿营地到出发点的距离依次为 $x_1,x_2,\cdots,x_n$,且满足 $x_1<x_2<\cdots<x_n$. 每天他们只能前进 30 千米,而任意两个相邻的宿营地之间的距离都不超过 30 千米. 在每个宿营地只住 1 天. 他们希望找到一个行动计划,使得总的宿营天数达到最少. 设计一个算法求解这个问题. 给出算法的主要步骤,证明算法是正确的,估计算法的时间复杂度.

4.9 有 n 个进程 $p_1,p_2,\cdots,p_n$. 对于 $i=1,2,\cdots,n$,进程 i 的开始时间为 $s[i]$,截止时间为 $d[i]$. 可以通过监测程序 Test 来测试正在运行的进程. Test 每次测试的时间很短,可以忽略不计. 换句话说,如果 Test 在时刻 t 进行测试,那么它将对满足 $s[i]\leqslant t\leqslant d[i]$的所有进程 p_i 同时取得测试数据. 假设最早运行的进程的开始时刻是 0,问: 如何安排测试时刻,使得对每个进程至少测试一次,且 Test 测试的次数达到最少? 说明算法的主要设计思想,给出伪码,证明算法的正确性,并分析算法最坏情况下的时间复杂度.

4.10 考虑习题 3.11 中关于圆环上 n 个排序数组的归并问题. 假设 n 个排序数组分别含有整数 $x_0,x_1,\cdots,x_{n-1}$个,如下设计贪心法: 计算所有相邻两个数组的元素数之和,从中选择元素数之和最小的两个数组进行归并. 这种贪心法是否能够对所有的实例得到最优解? 证明你的结果.

4.11 Dijkstra 算法要求有向图的边的权是非负实数. 请举出反例说明,对于某些含有负数边权的有向图,Dijkstra 算法不能得到正确的解.

4.12 设字符集 S,其中 8 个字符 A,B,C,D,E,F,G,H 的频率是 $f_1,f_2,\cdots,f_8$,且 $100\times f_i$ 是第 i 个 Fibonacci 数的值,$i=1,2,\cdots,8$.

(1) 给出这 8 个字符的 Huffman 树和编码.

(2) 如果有 n 个字符,其频率恰好对应前 n 个 Fibonacci 数,那么 Huffman 树是什么结构,证明你的结论.

4.13 设有作业集合 $J=\{1,2,\cdots,n\}$,每项作业的加工时间都是 1. 所有作业的截止时间是 D. 若作业 i 在 D 之后完成,则称为被延误的作业,并需要赔偿罚款 $m(i)$. 这里的 D 和 $m(i)(i=1,2,\cdots,n)$都是正整数,且 n 项 $m(i)$彼此不等. J 的一个调度是函数 f: $J\to\mathbf{N}$,其中 $\mathbf{N}$ 为自然数集合,$f(i)$表示作业 i 开始加工的时间,$i=1,2,\cdots,n$. 设计一个算法求出使得总罚款最少的调度,证明算法的正确性并给出最坏情况下的时间复杂度.

4.14 设有作业集合 $J=\{1,2,\cdots,n\}$，每项作业的加工时间都是 1. 作业 i 的截止时间是 $d(i)$，在 $d(i)$ 之前完成则获得利润 $m(i)$. 这里的 $d(i)$ 和 $m(i)(i=1,2,\cdots,n)$ 都是正整数，且所有的 $m(i)$ 彼此不等. J 的一个调度是函数 $f: J\to \mathbf{N}$，其中 $\mathbf{N}$ 为自然数集合，$f(i)$ 表示作业 i 开始加工的时间，$i=1,2,\cdots,n$. 设计一个算法求出使得总利润最大的调度，证明算法的正确性并给出最坏情况下的时间复杂度.

4.15 有 n 项任务的集合 $T=\{1,2,\cdots,n\}$，每项任务需要先放到机器 A 上进行预处理，然后再放到机器 B 上加工. 第 $i(i=1,2,\cdots,n)$ 项任务的预处理和加工时间分别是 $a(i)$ 和 $b(i)$，如果机器 A 只有 1 台，机器 B 的数量不限，问如何安排这些任务在机器 A 上的处理顺序，以使得总的加工时间最短？总加工时间的含义是：从 0 时刻机器 A 开始预处理，到 t 时刻最后一台机器 B 停止工作，总加工时间就是 t. 给出求解该问题的算法，用文字说明算法的主要设计思想和最坏情况下的时间复杂度，证明算法的正确性.

4.16 设 $A=\langle a_1,a_2,\cdots,a_n\rangle$，$B=\langle b_1,b_2,\cdots,b_m\rangle$ 是两个序列，其中 $m\leqslant n$. 设计一个 $O(n)$ 时间的算法，判断 B 是否为 A 的子序列. 说明算法的设计思想，给出伪码并证明算法的正确性.

4.17 设 $G=\langle V,E,W\rangle$ 是一个通信网络，其中结点集 V 是站点集合，边集 E 是站点之间的链路集合，$\forall e\in E$，权值 $w(e)$ 表示带宽，并且假设每条边的权都不相等. 对于任意站点 $u,v\in V$，一条 u-v 路径 P 的最大带宽是 $w(P)=\min\limits_{e\in P}\{w(e)\}$，即这条路径上的所有边的带宽的最小值. 而 u-v 之间的最佳带宽 $w(u,v)=\max\{w(P)\mid P$ 是一条 u-v 路径$\}$，即所有 u-v 路径带宽的最大值. 这也是 u 与 v 之间通信的最佳带宽.

(1) 证明存在一棵生成树，使得在这棵树中，连接每对结点 u 和 v 唯一路径的最大带宽等于 u 与 v 之间的最佳带宽.

(2) 设计一个找这样一棵生成树的算法，并分析算法的时间复杂度.

4.18 设 $S=\{1,2,\cdots,n\}$ 是 n 项广告的集合，广告 $i(i=1,2,\cdots,n)$ 有发布开始时间 $s(i)$ 和截止时间 $d(i)$，发布效益是 $v(i)$，其中 $s(i)$ 是非负整数，$d(i)$ 和 $v(i)$ 是正整数，且 $d(1)\leqslant d(2)\leqslant\cdots\leqslant d(n)$. 问题是：如何在 S 中选择一组广告 A，使得 A 中任意两个广告都相容(时间段不重叠)且总效益最大？

(1) 假设所有广告的效益都相等，试设计一个求解上述问题的算法，证明其正确性，并说明时间复杂度.

(2) 如果效益 $v(i)$ 可以取任意正整数，设计一个算法求解这个问题，用文字说明算法的设计思想和主要步骤，分析算法最坏情况下的时间复杂度.

4.19 有 n 个文件存在磁带上，每个文件占用连续的空间. 已知第 i 个文件需要的存储空间为 s_i，被检索的概率是 f_i，$i=1,2,\cdots,n$，且 $f_1+f_2+\cdots+f_n=1$. 检索每个文件需要从磁带的开始位置进行操作，比如文件 i 需要空间 $s_i=310$，存储在磁带的 121～430 单元，那么检索该文件需要的时间为 430. 问如何排列 n 个文件使得平均检索时间最少？设计算法求解这个问题，说明算法的设计思想，证明算法的正确性，给出算法最坏情况下的时间复杂度.

4.20 一个公司需要购买 n 个密码软件的许可证，按照规定每个月至多可以得到一个软件的许可证. 假定每个许可证目前售价都是 1000 元. 但是第 i 个许可证的售价将按照 $r_i>1$ 的指数因子增长，$i=1,2,\cdots,n$. 例如，第 i 个许可证在 1 个月后将是 $r_i\times 1000$ 元，2 个

月后将是 $r_i^2\times 1000$ 元,k 个月后将是 $r_i^k\times 1000$ 元. 给定输入 $r_1, r_2, \cdots, r_n$,给出一个购买许可证的顺序,以使得花费的总钱数最少. 设计一个算法求解这个问题,说明设计思想,证明其正确性并分析算法最坏情况下的时间复杂度.

4.21 给定 n 个集合 $A_1, A_2, \cdots, A_n$,每个集合都由连续的正整数构成,即

$$A_i = \{x \mid a_i \leqslant x \leqslant b_i\},\quad a_i, x, b_i \in \mathbf{Z}^+,\ i = 1,2,\cdots,n$$

设计一个算法求最小的集合 S,使得对每个 $i=1,2,\cdots,n$, $|S\cap A_i|\neq\varnothing$,即每个 A_i 至少含有 S 中的一个数.

4.3 习题解答与分析

4.1 (1) 假设调度 f 的顺序是 $i_1, i_2, \cdots, i_n$,那么 i_k 的等待时间是 $\sum_{j=1}^{k-1} t_{i_j}$. 总的等待时间

$$time(f) = \sum_{i=1}^{n}(n-i)t_{i_j}$$

贪心策略是:服务时间较短的优先安排. 排序使得 $t_1\leqslant t_2\leqslant\cdots\leqslant t_n$,按照 $1,2,\cdots,n$ 的顺序安排服务. f^* 的总等待时间为

$$time(f^*) = \sum_{i=1}^{n}(n-i)t_i$$

命题 4.1 对任何输入,对服务时间短的顾客优先安排将得到最优解.

证 使用交换论证的方法. 假设存在某个最优解 g 存在 $i, j\in A, i<j$,但是 i 在 j 之后得到服务. 那么在 g 的安排中一定存在相邻安排的顾客 i 和 j,使得 $i<j$ 但 i 在 j 之后得到服务. 交换 i 和 j 得到新的服务顺序 g'. 那么

$$time(g) - time(g') = t_j - t_i \geqslant 0$$

总等待时间将不会增加,因此 g' 也是最优解. 至多经过 $n(n-1)/2$ 次这样的交换,就可以将 g 转化成算法的解 f^*. 从而证明了 f^* 是最优解.

(2) 算法 Service(T).

输入: $T[1..n]$ 是服务时间 $t_1, t_2, \cdots, t_n$ 的数组

输出: 函数 f, $f(i)$ 为第 i 个顾客的开始服务时刻, $i=1,2,\cdots,n$

```
1.  sort(T)                        //按照服务时间从小到大的顺序排列
2.  f(1)←0
3.  for i←2 to n
4.      f(i)←f(i−1)+t_{i−1}
5.  return f
```

算法的最坏情况下的时间复杂度为 $O(n\log n)$.

(3) 服务顺序为: 1,3,2,6,5,7,4. 其开始服务时刻为: $f(1)=0, f(3)=1, f(2)=3, f(6)=6, f(5)=12, f(7)=22, f(4)=34$. 总等待时间

$$Time(f) = 1\times 6 + 2\times 5 + 3\times 4 + 6\times 3 + 10\times 2 + 12\times 1 = 78$$

4.2 使用贪心法.

贪心策略：按照底面长度从小到大对物品排序，从标号小的物品开始依次装入库房，直到某个物品装不下为止.

命题 4.2 对任何输入，把物品按照从小到大的顺序装入将得到最优解.

证 使用交换论证的方法. 令物品集合为 $S=\{1,2,\cdots,n\}$. 假设 $A=\{i_1,i_2,\cdots,i_k\}$是一个最优解，不妨设 $i_1<i_2<\cdots<i_k$. 与上述算法的解 $B=\{1,2,\cdots\}$比较，假若 $i_1=1,i_2=2,\cdots,i_{j-1}=j-1,i_j\neq j$，那么把 i_j 替换成 j，即

$$A'=(A-\{i_j\})\cup\{j\}$$

由于 $i_j>j$，其底面长度不小于 j 的底面长度，A'是一个可行解；且由于$|A'|=|A|$，A'也是一个最优解. 如上不断替换 A，至多经过 $n-1$ 次就可以得到 B，从而证明 B 是一个最优解.

算法最坏情况下的时间复杂度是 $O(n\log n)$.

4.3 使用贪心法. 令 $a_1,a_2,\cdots$表示基站的位置.

贪心策略：首先令 $a_1=d_1+4$. 对 $d_2,d_3,\cdots,d_n$ 依次检查，找到下一个不能被该基站覆盖的房子. 如果 $d_k\leqslant a_1+4$ 但 $d_{k+1}>a_1+4$，那么第 $k+1$ 个房子不能被基站覆盖，于是取 $a_2=d_{k+1}+4$ 作为下一个基站的位置. 照此下去，直到检查完 d_n 为止.

算法的伪码如下：

Location

输入：距离 $d_1,d_2,\cdots,d_n$ 的数组 $d[1..n]$，满足 $d[1]<d[2]<\cdots<d[n]$

输出：基站位置的数组 a

```
a[1]←d[1]+4
k←1
for j←2 to n
    if d[j]>a[k]+4
    then k←k+1
        a[k]←d[j]+4
return a
```

算法正确性证明使用归纳法.

命题 4.3 对任何正整数 k，存在最优解包含算法前 k 步选择的基站位置.

证 $k=1$，存在最优解包含 $a[1]$. 若不然，有最优解 OPT，其第一个位置是 $b[1]$，$b[1]\neq a[1]$，那么 $d_1-4\leqslant b[1]<d_1+4=a[1]$. $b[1]$覆盖的是距离在$[d_1,b[1]+4]$之间的房子. $a[1]$覆盖的是距离在$[d_1,a[1]+4]$的房子. 因为 $b[1]<a[1]$，$b[1]$覆盖的房子都在 $a[1]$覆盖的区域内，用 $a[1]$替换 $b[1]$，得到的仍旧是最优解.

假设对于 k，存在最优解 A 包含算法前 k 步选择的基站位置，即

$$A=\{a[1],a[2],\cdots,a[k]\}\cup B$$

其中 $a[1],a[2],\cdots,a[k]$覆盖了距离 $d_1,d_2,\cdots,d_j$ 的房子. 那么，B 是关于 $L=\{d_{j+1},d_{j+2},\cdots,d_n\}$的最优解. 否则，存在关于 L 的更优的解 B^*，那么用 B^* 替换 B 就得到 A^*，且$|A^*|<|A|$，与 A 的最优性矛盾. 根据归纳基础，L 有一个最优解 $B'=\{a[k+1],\cdots\}$，$|B'|=|B|$. 于是

$$\begin{aligned}A'&=\{a[1],a[2],\cdots,a[k]\}\cup B'\\&=\{a[1],a[2],\cdots,a[k],a[k+1],\cdots\}\end{aligned}$$

且$|A'|=|A|$,A'也是最优解. 从而证明了命题对$k+1$也为真. 根据归纳法,对任何正整数k命题都成立.

第 3 行的 for 循环运行$O(n)$次,循环体内操作为常数时间,因此算法最坏情况下的时间复杂度是$O(n)$.

4.4 使用贪心法.

贪心策略:从x_1取起. 第一个区间是$[x_1,x_1+1]$. 顺序考察后面的点,假设最后一个落入该区间的点是x_k,即$x_k\leqslant x_1+1$,$x_{k+1}>x_1+1$. 那么下一个区间是$[x_{k+1},x_{k+1}+1]$. 按照这样的办法直到剩下的所有的点都落入最后一个区间为止.

算法正确性证明使用归纳法.

命题 4.4 对任何正整数j,算法进行到第j步,得到第 1 到第j个区间,一定存在一个最优解包含这j个区间.

证 对j归纳. 下面使用符号$\tau(a)$表示单位区间$[a,a+1]$.

$j=1$,即证明存在最优解包含了区间$\tau(x_1)$.

x_1点一定在任何最优解的某个区间中,而且在第一个区间里. 若不然,在包含x_1的区间左边还有别的区间,那么这些区间不可能包含X中的点,因此这些区间可以从解中去掉,与解的最优性矛盾. 设最优解的第一个区间是$\tau(a)$,假设$a<x_1$,那么用$\tau(x_1)$替换$\tau(a)$,$\tau(x_1)$完全覆盖了$\tau(a)$中的点,因此替换后仍旧是最优解.

$j=k$,假设存在最优解T包含算法前k步选择的k个区间,即

$$T=\{\tau(x_{i_1}),\tau(x_{i_2}),\cdots,\tau(x_{i_k})\}\cup T'$$

其中$x_{i_1},x_{i_2},\cdots,x_{i_k}$是算法从$X$中选择的$k$个区间的左端点,且$i_1=1$. 设这$k$个区间包含了点$x_1,x_2,\cdots,x_p$. 那么考虑后面$n-p$个点$x_{p+1},\cdots,x_n$的区间选择子问题,而$T'$是子问题$X'=\{x_{p+1},x_{p+2},\cdots,x_n\}$的一个最优解. 如若不然,$X'$有着更好的解$T''$,那么用$T''$替换$T'$,就可以得到原问题的一个比$T$更好的解,这与归纳假设矛盾. 考虑子问题$X'$. 由归纳基础,对于$X'$存在一个最优解$T_1=\{\tau(x_{p+1})\}\cup T_2$,下面证明

$$T^*=\{\tau(x_{i_1}),\tau(x_{i_2}),\cdots,\tau(x_{i_k})\}\cup T_1$$

也是原问题的最优解. 假设T^*不是最优解,那么T_1比T'的区间个数至少多 1,因此与T_1是$X'=\{x_{p+1},x_{p+2},\cdots,x_n\}$的最优解矛盾. 于是得到

$$\begin{aligned}T^*&=\{\tau(x_{i_1}),\tau(x_{i_2}),\cdots,\tau(x_{i_k})\}\cup\{\tau(x_{p+1})\}\cup T_2\\&=\{\tau(x_{i_1}),\tau(x_{i_2}),\cdots,\tau(x_{i_k}),\tau(x_{p+1})\}\cup T_2\\&=\{\tau(x_{i_1}),\tau(x_{i_2}),\cdots,\tau(x_{i_k}),\tau(x_{i_{k+1}})\}\cup T_2\end{aligned}$$

从而算法前$k+1$步的选择也可以导致最优解. 根据数学归纳法,命题得证.

算法最坏情况下的时间复杂度为$O(n)$.

4.5 (1) 使用贪心法.

贪心策略:将文件按照所需存储空间从小到大的次序排列,将排序后的标号记作 1,2,…,n,即$p_1\leqslant p_2\leqslant\cdots\leqslant p_n$. 然后按照 1,2,…,$n$的文件排列次序将它们依次存入磁盘.

命题 4.5 对任何输入,按照文件所需空间从小到大存入磁盘将得到最优解.

证 使用交换论证. 不难看出,只要存放文件名称相同(不管次序)的任何方法都是同样的解. 不妨设最优解为$\mathrm{OPT}=\{i_1,i_2,\cdots,i_j\}$,$i_1<i_2<\cdots<i_j$,$j\leqslant n$. 如果

$$\{i_1,i_2,\cdots,i_j\}=\{1,2,\cdots,j\}$$

那么算法的解就是最优解. 假设$\{i_1,i_2,\cdots,i_j\}\neq\{1,2,\cdots,j\}$,设 $i_1=1,i_2=2,\cdots,i_{t-1}=t-1$, $i_t>t$. 用 t 替换 i_t,那么得到的解 I^* 占用的存储空间与 OPT 占用空间的差

$$S(I^*)-S(OPT)=p_t-p_{i_t}\leqslant 0,\quad 由于\ t<i_t$$

因此 I^* 也是最优解,但是它比 OPT 减少了一个标号不相等的文件. 对于解 OPT,从第一个标号不等的文件开始,经过至多 j 次替换,就得到最优解$\{1,2,\cdots,j\}$.

算法最坏情况下的时间复杂度为

$$W(n)=O(n\log n)+O(n)=O(n\log n)$$

(2) 如果要求剩余空间达到最小,这个问题本质上是 0-1 背包问题,每个文件相当于物品,其重量和价值就是所需存储空间,背包的重量限制等于磁盘容量 C. 可以使用动态规划的设计技术.

设 $F_k(y)$表示考虑前 k 个文件,磁盘空间为 y 时的最大存储量. 递推方程是

$$F_k(y)=\begin{cases}\max\{F_{k-1}(y),F_{k-1}(y-p_k)+p_k\}, & p_k\leqslant y\leqslant C\\ F_{k-1}(y), & p_k>y\end{cases},\quad k>0$$

$$F_0(y)=0,\quad 0\leqslant y\leqslant C$$

$$F_k(0)=0$$

$$F_k(y)=-\infty,\quad y<0$$

如下设定标记函数 $i_k(y)$用于追踪解.

$$i_k(y)=\begin{cases}k & 若\ F_{k-1}(y)\leqslant F_{k-1}(y-p_k)+p_k,p_k\leqslant y\leqslant C\\ i_{k-1}(y) & 否则\end{cases},\quad k>1$$

$$i_1(y)=\begin{cases}1 & 若\ y\geqslant p_1\\ 0 & 否则\end{cases}$$

算法最坏情况下的时间复杂度是 $O(nC)$.

4.6 使用贪心法.

贪心策略:按照加工时间从小到大对作业排序,使得 $t(i)\leqslant t(i+1),i=1,2,\cdots,n-1$. 按照标号从小到大的次序安排所有的作业.

命题 4.6 对任何输入,按照加工时间从小到大安排任务将得到最优解.

证 使用交换论证的方法. 任意给定最优解 $f=<i_1,i_2,\cdots,i_n>$,不妨设从 0 时刻计时,且 f 的安排中没有空闲时间. 任务 i_j 的完成时间是:

$$time(i_j)=f(i_j)+t(i_j)=\sum_{k=1}^{j-1}t(i_k)+t(i_j)=\sum_{k=1}^{j}t(i_j)$$

f 的总完成时间是

$$time(f)=\sum_{k=1}^{n}time(i_k)=\sum_{j=1}^{n}t(i_j)(n-j+1)$$

由于平均完成时间为$\frac{1}{n}time(f)$,当 $time(f)$取得最小值时得到最优解 f.

假如在 f 中存在逆序(i_j,i_k)使得 $j<k$ 但是 $i_j>i_k$,必有相邻的逆序. 交换具有逆序的相邻元素 i_j 与 i_k,得到解 g. 那么总的完成时间之差是

$$time(g)-time(f)=t(i_k)-t(i_j)\leqslant 0$$

因此 g 也是最优解. 至多经过 $n(n-1)/2$ 次交换,去除所有的逆序,就可以使得 f 转变成贪心法的解,从而证明了贪心法的解是最优解.

算法在最坏情况下的时间复杂度为 $O(n\log n)$.

4.7 使用贪心法.

贪心策略:按币值从大到小排列零钱,从币值大的开始,每种钱尽量多用. 如果剩余钱数小于该币值,再考虑用下一种钱币. 可以用两种方法证明算法的正确性.

方法一:使用"一点定理"(主教材定理4.6).

$n=1$,只有1种钱,贪心法对任意 y 显然得到最优解.

假设对于任意 n 种币值,贪心法都得到最优解,考虑 $n+1$ 种币值 $\{1,p,p^2,\cdots,p^n\}$,由

$$w_{n-1}=w_n=1, p^n=p^{n-1}\cdot p-\delta$$

得 $\delta=0$,于是

$$w_n+G_{n-1}(\delta)=1<p=pw_{n-1}$$

成立,根据一点定理,贪心法对 $n+1$ 种币值的钱币系统也得到最优解.

方法二:先证明以下命题.

命题 4.7 设币值系统是 $\{1,p,p^2,\cdots,p^n\}$. 假设 $p^j\leqslant y<p^{j+1}$,那么钱数等于 y 的最优解含有 $t=y/p^j$ 个币值 p^j 的钱币.

证 假若最优解不含币值 p^j 的钱币,即

$$y=x_0+x_1p+x_2p^2+\cdots+x_{j-1}p^{j-1}$$

那么存在 $x_i\geqslant p, i\in\{0,1,\cdots,j-1\}$. 如果不是,则

$$y=x_0+x_1p+x_2p^2+\cdots+x_{j-1}p^{j-1}\leqslant(p-1)(1+p+p^2+\cdots+p^{j-1})=p^j-1$$

与 $y\geqslant p^j$ 矛盾. 不妨设 $x_i\geqslant p$,那么用1个币值为 p^{i+1} 的钱币替换 p 个币值为 p^i 的钱币,总钱币数将减少 $p-1$,与这个解为最优解矛盾.

下面证明最优解恰好含有 y/p^j 个币值 p^j 的钱币. 设钱数为 y 时最优解是 $F_j(y)$,那么

$$F_j(y)=\begin{cases}F_j(y-p^j)+1 & y\geqslant p^j\\ F_{j-1}(y) & y<p^j\end{cases}$$

设 $y=tp^j+\delta(\delta<p^j)$,其中 $t=y/p^j$. 通过对 t 的归纳不难证明 $F_j(y)=t+F_{j-1}(\delta)$,即最优解中含有 y/p^j 个币值 p^j 的钱币.

不难看到上述解就是贪心法的解. 根据命题4.7,这个解是最优解.

上述算法在最坏情况下的时间复杂度是 $W(n)=O(\min\{n,\log y\})$.

4.8 使用贪心法.

贪心策略:每天尽可能选择距离出发点不超过30千米的最远的宿营地.

算法 Plan

1. 先找第一个距出发点小于等于30千米,且距出发点最远的宿营地 a_1
2. $i\leftarrow a_1$
3. if i 距终点>30
4. then 从 i 出发,找距 i 不超过30千米,且距 i 最远的宿营地 j
5. $i\leftarrow j$,转步骤3

算法的正确性证明.

命题 4.8 对于任何 k,算法进行到第 k 步,找到 k 个宿营地 $x_{i_1},x_{i_2},\cdots,x_{i_k}$,一定存在最优解 T 包含 $x_{i_1},x_{i_2},\cdots,x_{i_k}$,且 $|T|$ 最小.

证 使用归纳法. 设出发点为 s,终点为 t.

$k=1$,即证明存在一个最优解选的第一个营地是 x_{i_1}. 假设最优解 f 的第一个宿营地为 x_j,不是 x_{i_1}. 由于 x_{i_1} 比 x_j 距 s 更远,因此 x_{i_1} 距 f 的下一个宿营地距离不超过 30 千米,且 x_{i_1} 距 s 也不超过 30 千米,那么令

$$T'=(T-\{x_j\})\cup\{x_{i_1}\}$$

则 $|T'|=|T|$,且 T' 也是一个最优解.

假设算法到第 k 步找到 k 个宿营地 $x_{i_1},x_{i_2},\cdots,x_{i_k}$,且存在最优解 T 包含 $x_{i_1},x_{i_2},\cdots,x_{i_k}$. 剩下的是起点为 x_{i_k}、终点不变的子问题 P_1,根据归纳基础,对子问题 P_1 存在最优解 T_1 包含 x_p,其中 $x_p\leqslant x_{i_k}+30$ 且是距 x_{i_k} 最远的宿营地,即 $T_1=\{x_p\}\cup T_2$,根据算法有 $p=i_{k+1}$. 下面证明 $\{x_{i_1},x_{i_2},\cdots,x_{i_k}\}\cup T_1$ 也是原问题的最优解. 若不然,存在最优解 $T^*=\{x_{i_1},x_{i_2},\cdots,x_{i_k}\}\cup T_1'$,$T_1'$ 是 P_1 的最优解且 $|T^*|<|T_1|+k$,那么 $|T_1'|<|T_1|$,与 T_1 是子问题 P_1 的最优解矛盾. 于是

$$\{x_{i_1},x_{i_2},\cdots,x_{i_k}\}\cup T_1=\{x_{i_1},x_{i_2},\cdots,x_{i_k},x_{i_{k+1}}\}\cup T_2$$

是原问题的最优解,从而证明了算法第 $1,2,\cdots,k,k+1$ 步的选择也导致最优解.

算法在最坏情况下的时间复杂度是 $W(n)=O(n)$.

4.9 使用贪心法.

贪心策略:把进程按截止时间排序. 取第一个进程的截止时间作为第一个测试点,然后顺序检查后续能够被这个测试点检测的进程(这些进程的开始时间小于等于测试点),直到找到下一个不能被测试到的进程为止. 取这个进程的截止时间作为下一个测试点,……,直到检查完所有的进程为止. 伪码如下:

Test

输入:$s[1..n]$,$d[1..n]$

输出:t,顺序选定的测试点构成的数组

1. 将进程按照 $d[i]$ 递增的顺序排序,使得 $d[1]\leqslant d[2]\leqslant\cdots\leqslant d[n]$
2. $i\leftarrow 1$, $t[i]\leftarrow d[1]$ //第一个测试点是最早结束进程的截止时间
3. $j\leftarrow 2$
4. while $j\leqslant n$ and $s[j]\leqslant t[i]$ do //检查进程 j 是否可以在时刻 $t[i]$ 被测试
5. $\quad j\leftarrow j+1$
6. if $j>n$ then return t
7. else $i\leftarrow i+1$ //找到待测进程中结束时间最早的进程 j
8. $\quad t[i]\leftarrow d[j]$
9. $\quad j\leftarrow j+1$, goto 4

下面通过对步数 k 归纳证明算法的正确性.

命题 4.9 对于任何 k,存在最优解包含算法前 k 步选择的测试点.

证 $k=1$,设 $S=\{t[i_1],t[i_2],\cdots\}$ 是最优解,不妨设 $t[i_1]<t[1]$. 设 p_u 是在时刻 $t[i_1]$ 被测到的任意进程,那么 $s(u)\leqslant t[i_1]\leqslant d[u]$,从而有

$$s[u] \leqslant t[i_1] < t[1] = d[1] \leqslant d[u]$$

因此,p_u 也可以在 $t[1]$时刻被测试. 于是在 S 中用 $t[1]$替换 $t[i_1]$后也得到一个最优解.

假设对于任意 k,算法在前 k 步选择了 k 个测试点 $t[1],t[i_2],\cdots,t[i_k]$,且存在最优解 $T=\{t[1],t[i_2],\cdots,t[i_k]\}\cup T'$. 设算法前 k 步选择的测试点不能测到的进程构成集合 $Q\subseteq P$,其中 P 为全体进程的集合. 不难证明 T'是子问题 Q 的最优解. 根据归纳基础,存在 Q 的最优解 T^* 包含测试点 $t[i_{k+1}]$,即

$$T^* = \{t[i_{k+1}]\}\cup T''$$

因此

$$\{t[1],t[i_2],\cdots,t[i_k]\}\cup T^* = \{t[1],t[i_2],\cdots,t[i_{k+1}]\}\cup T''$$

也是原问题的最优解. 根据归纳法,命题正确.

算法最坏情况下的时间复杂度是 $W(n)=O(n\log n)$.

4.10 对某些实例不能得到最优解. 反例如下:

设排序序列是 A_0,A_1,A_2,A_3,A_4;含有的整数个数分别为 $x_0=2,x_1=3,x_2=10,x_3=3,x_4=2$. 一个最优的归并次序是:

1. A_0 与 A_1 归并得到 A_{01},最坏情况下的比较次数是 $2+3-1=4$.
2. A_3 与 A_4 归并得到 A_{34},最坏情况下的比较次数是 $2+3-1=4$.
3. A_{34} 与 A_{01} 归并得到 A_{31},最坏情况下的比较次数是 $5+5-1=9$.
4. A_{31} 与 A_2 归并,最坏情况下的比较次数是 $10+10-1=19$.

总计需要的比较次数是

$$4+4+9+19=36$$

而贪心法的归并次序是:

1. A_4 与 A_0 归并得到 A_{40},最坏情况下的比较次数是 $2+2-1=3$.
2. A_{40} 与 A_1 归并得到 A_{41},最坏情况下的比较次数是 $4+3-1=6$.
3. A_3 与 A_{41} 归并得到 A_{31},最坏情况下的比较次数是 $3+7-1=9$.
4. A_{31} 与 A_2 归并,最坏情况下的比较次数是 $10+10-1=19$.

总计需要的比较次数是

$$3+6+9+19=37$$

4.11 解 如图 4.1 所示,设 $G=\langle V,E,W\rangle$,运行 Dijkstra 算法.

第 1 步,$S=\{s\}$,$dist[1]=1,dist[2]=dist[3]=dist[4]=\infty$.

第 2 步,$S=\{s,1\}$,修改 $dist[2]=0$.

第 3 步,$S=\{s,1,2\}$,修改 $dist[3]=-2$.

第 4 步,$S=\{s,1,2,3\}$,修改 $dist[4]=2$.

按照算法,$short[1]=1$. 但是因为存在负圈 $1\to2\to3\to1$,从 s 到 1 的路径可以沿着负圈走下去. 每走一圈,路径长度减少 -6,这个过程不会终止,因此图中的最短路径实际上是不存在的.

4.12 (1) Huffman 树如图 4.2 所示. 编码为

H:0, G:10, F:110, E:1110, D:11110,
C:111110, B:1111110, A:1111111

(2) 先证明以下命题.

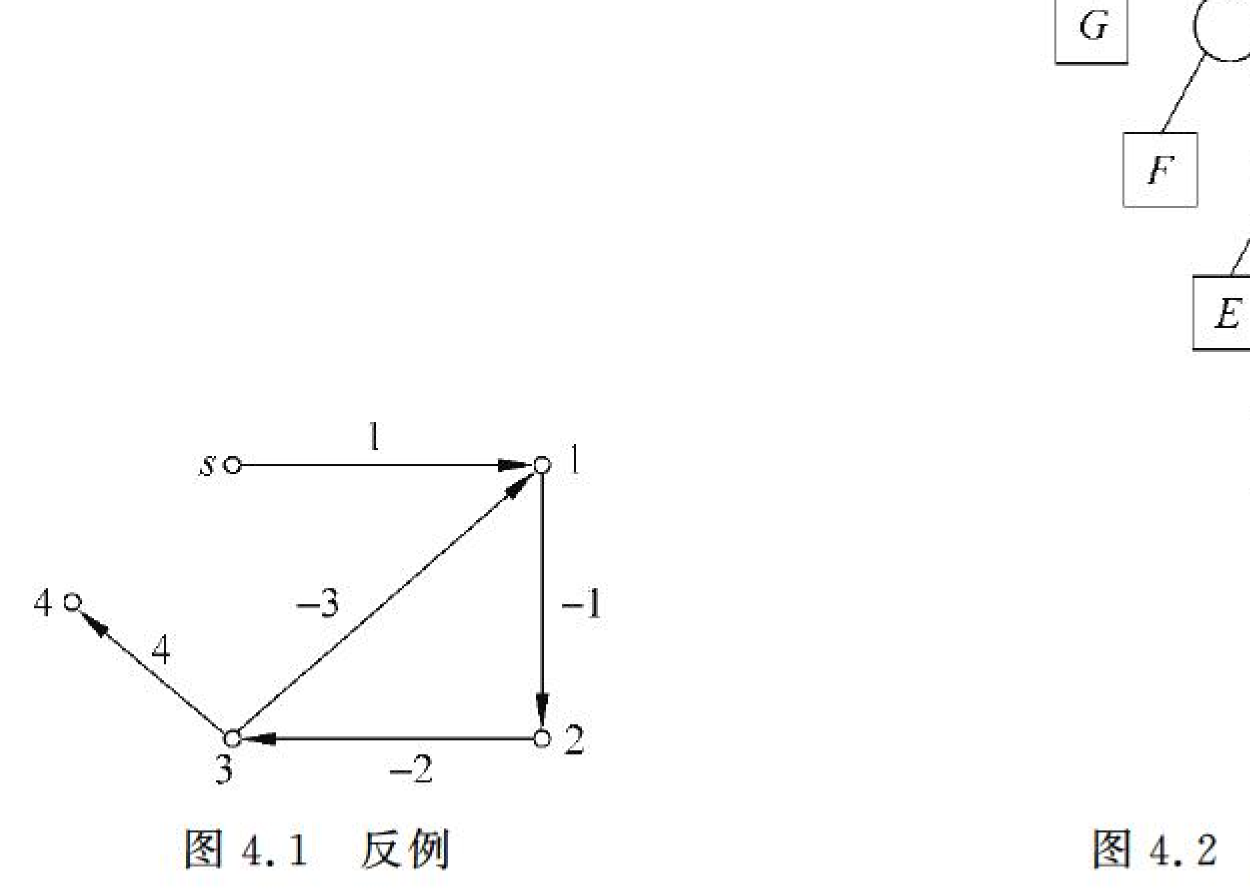


图 4.1　反例

H
G
F
E
D
C
B
A

图 4.2　Huffman 树

命题 4.10　设 $f_1, f_2, \cdots$ 为 Fibonacci 数列，则

$$\sum_{i=1}^{k} f_i \leqslant f_{k+2}$$

证　对 k 归纳.

当 $k=1$ 时，$f_1 < f_3$ 显然为真.

假设 $k=n$ 时命题成立，则 $k=n+1$ 时，有

$$\sum_{i=1}^{n+1} f_i = \sum_{i=1}^{n} f_i + f_{n+1} \leqslant f_{n+2} + f_{n+1} = f_{n+3}$$

所以命题对于 $k=n+1$ 成立.

综上所述，$\sum_{i=1}^{k} f_i \leqslant f_{k+2}$ 对任意正整数 k 成立.

根据命题 4.10，前 k 个字符合并后子树的根的权值不大于第 $k+2$ 个 Fibonacci 数. 根据 Huffman 算法，它将继续参加与第 $k+1$ 个字符的合并. 因此 n 个字符的 Huffman 编码按照频率从小到大依次是：

11…1(含 $n-1$ 个 1)，　11…10(含 $n-2$ 个 1)，　11…10(含 $n-3$ 个 1)，…，　10，　0

即第 $i(i>1)$ 个字母的编码为 11…10(含 $n-i$ 个 1).

4.13　使用贪心法.

贪心策略：优先安排前 D 个罚款最多的作业.

算法的正确性证明使用交换论证的方法. 先给出以下命题.

命题 4.11　设作业调度 f 的安排次序是 $\langle i_1, i_2, \cdots, i_n \rangle$，那么罚款

$$F(f) = \sum_{k=D+1}^{n} m(i_k)$$

证　显然最优调度没有空闲时间，不妨假设作业是连续安排的. 因为每项作业的加工时间都是 1，在截止时间 D 之前可以完成 D 项作业. 只有在 D 之后安排的 $n-D$ 项作业，即

作业 $i_{D+1},i_{D+2},\cdots,i_n$ 是被罚款的作业.

从命题4.11可以直接推出以下结果：令 S 是 $n-D$ 项罚款最少的作业构成的集合.

(1) 对于 S 中的作业 i 和 j，或对于 $J-S$ 中的作业 i 和 j，交换 i 和 j 的加工顺序不影响总罚款.

(2) 对于作业 i 和 j，$m(i)<m(j)$，调度 f 将 i 安排在 D 之前，j 安排在 D 之后，那么交换作业 i 和 j，得到调度 g，则 g 的罚款将减少. 因为

$$F(g)-F(f)=m(i)-m(j)<0$$

根据上述分析，不难看到，把罚款最小的 $n-D$ 项作业安排在最后将使得罚款总额达到最小. 设计以下算法：

1. 利用 Select 算法从 $m(1),m(2),\cdots,m(n)$ 中选第 $n-D$ 小，记作 m^*.

2. 用 m^* 与其他 $n-1$ 个 $m(i)$ 进行比较，找出比 m^* 小的 $n-D-1$ 个 $m(i)$.

3. 将上述 $n-D$ 项 $m(i)$（含 m^* 在内）对应的作业从 D 时刻开始以任意顺序安排加工.

4. 将剩下的 D 项作业以任意顺序安排在 $0,1,\cdots,D-1$ 时刻加工.

上述算法本质上属于贪心法. 根据前面的分析，它的总罚款数与完全按照 $m(i)$ 从大到小安排作业的贪心法是一样的. 但是贪心法需要对作业按照 $m(i)$ 的大小排序，最坏情况下的时间复杂度是 $O(n\log n)$；采用上述基于 Select 选择过程的算法需要的时间不超过 $O(n)$.

4.14 设 f 是一个调度，其不被延迟的作业集合是 $A(f)\subseteq\{1,2,\cdots,n\}$，那么 f 的利润是

$$T(f)=\sum_{i\in A(f)}m(i)$$

$$A(f)=\{i\mid i\in J,f(i)+1\leqslant d(i)\}$$

贪心策略：先把作业按照利润从大到小排序，使得 $m(1)>m(2)>\cdots>m(n)$. 然后从第1项到第 n 项依次做出安排. 对作业 $i(i=1,2,\cdots,n)$ 的安排方法是：

1. 如果 $d(i)-1$ 时刻没有安排作业，那么 $f(i)=d(i)-1$.

2. 如果 $d(i)-1$ 时刻已经安排了其他作业，那么从 $d(i)-1$ 位置沿时间区间 $[0,d[i]-1]$ 顺序向前搜索，找到机器的第一个空闲位置（即最晚的空闲位置）$[j,j+1]$，令 $f[i]=j$.

3. 如果从 $d(i)-1$ 直到时刻0机器没有空闲，那么将任务 i 放到集合 S 中.

4. 等到所有的作业按照上述方法扫描完毕之后，再以任意顺序安排 S 中的作业.

算法的正确性证明如下.

假设最优调度都没有空闲时间. 不难看到：对于两个不同的调度 f 和 g，它们对于某些作业安排的时间可能不一样. 但是，只要 $A(f)=A(g)$，一定有 $T(f)=T(g)$. 因此，在下面的证明中不再区分那些具有相同不延迟作业集的最优调度.

命题4.12 对于任意 k，算法执行到第 k 步，得到不延迟的作业安排 $f(1),f(i_2),\cdots,f(i_k)$，那么存在一个最优调度 g，使得对于 $m=1,2,\cdots,k$，有 $f(i_m)=g(i_m)$.

证 对 k 归纳.

$k=1$. 设 g 是一个最优解. 若作业1在 g 中被延迟，用作业1替换安排在 $d(1)-1$ 时刻的作业 j，得到 g'，那么 g' 的利润至少增加 $m(1)-m(j)>0$，与 g 的最优性矛盾. 若 $g(1)<d(1)-1$，顺序交换作业1和后续作业，直到 $g(1)=d(1)-1$ 为止. 显然与其交换的

作业都被提前，作业 1 保持利润不变，总利润不变. 因此交换后的 g 是最优解且 $f(1)=d(1)-1=g(1)$.

假设对于任何正整数 k 命题都为真，即存在一个最优调度 g，使得对于 $j=1,2,\cdots,k$，有 $f(i_j)=g(i_j)$，其中 $i_1=1$. 令 $f(i_{k+1})=t$.

1. 根据 f 的贪心策略，如果有作业 j，$i_k<j<i_{k+1}$，那么 j 属于被搁置的作业，在其截止时间 $d(j)$之前机器没有空闲. 由于 g 对作业 $i_1,i_2,\cdots,i_k$ 的安排与 f 相同，因此作业 j 在调度 g 中同样是被搁置的作业.

2. 如图 4.3 所示. 由于调度 g 没有空闲时间，那么存在作业 s 使得 $g(s)=t$. 根据上面的分析有 $s>i_{k+1}$. 假若 i_{k+1}在 g 中是被搁置的作业，那么由于 $m(i_{k+1})>m(s)$，将 s 替换成 i_{k+1}，g 的利润将增加，这与 g 的最优性矛盾，从而 i_{k+1}在 g 中只能被安排在 t 时刻之前. 交换作业 i_{k+1}和 s 的顺序，g 的利润不变，即交换后的 g 仍旧是最优解，且 $f(i_{k+1})=g(i_{k+1})$.

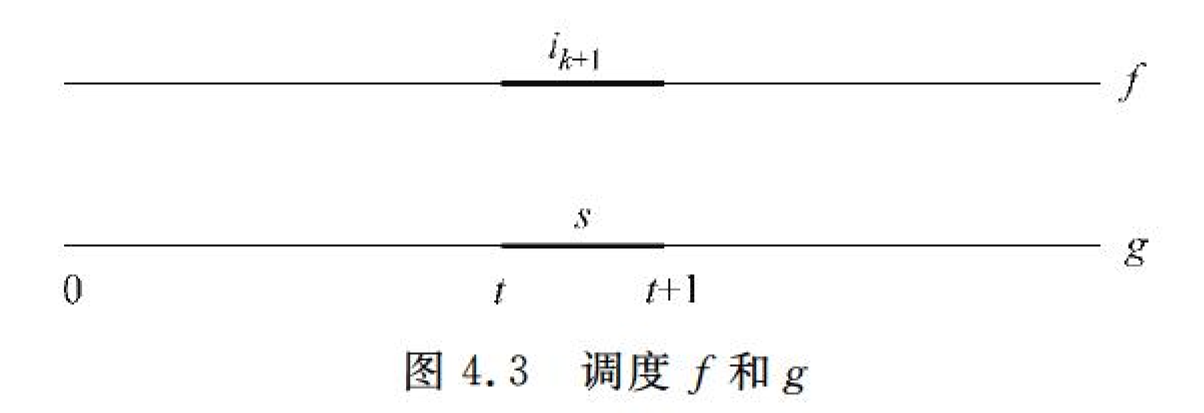


图 4.3　调度 f 和 g

根据归纳法，对于任何正整数 k，算法前 k 步的安排都将导致最优解.

算法在最坏情况下的时间复杂度是 $O(n\log n)$.

4.15　采用贪心法.

贪心策略：将任务按照 $b(i)$减少的顺序排序，使得 $b(1)\geqslant b(2)\geqslant\cdots\geqslant b(n)$，然后按照从 1 到 n 的顺序在机器 A 上安排预处理. 任何任务在 A 上完成预处理后，立刻放到一台闲置的机器 B 上进行加工.

在时刻 0 机器 A 开始工作，不妨设机器 A 没有空闲时间. 对于任何调度 $f=<i_1,i_2,\cdots,i_n>$，任务 $i_j(j=1,2,\cdots,n)$的完成时间是

$$d_f(i_j)=\sum_{k=1}^{j}a(i_k)+b(i_j)$$

总加工时间是

$$T(f)=\max\{d_f(i_j)\mid j=1,2,\cdots,n\}$$

下面使用交换论证的方法证明贪心法得到最优解.

命题 4.13　对于上述问题，选择在机器 B 上加工时间长的任务先做预处理的算法可得到最优解.

证　假设 $f=<i_1,i_2,\cdots,i_n>$是一个最优解，如果在 f 中存在逆序，即存在任务 j 和 k，$b(j)\leqslant b(k)$，但是 $f(j)<f(k)$. 不难看出，在 f 的所有逆序中一定存在连续安排的一对任务 j 和 k 构成逆序. 在 f 中交换任务 j 和 k 得到调度 g. 下面证明 g 也是最优调度.

首先不难看到：在调度 f 和 g 中，除了 j 与 k 之外的其他任务的完成时间都相等，因此影响 f 和 g 的总加工时间的只能是 j 和 k 的完成时间. 下面分别计算 j 和 k 在两个调度的完成时间.

如图 4.4 所示. 设 $f(j)=g(k)=C$，那么有

$$d_f(j) = C + a(j) + b(j)$$
$$d_f(k) = C + a(j) + a(k) + b(k)$$
$$d_g(k) = C + a(k) + b(k)$$
$$d_g(j) = C + a(k) + a(j) + b(j)$$

因为 $b(j) \leqslant b(k), a(j) > 0, a(k) > 0$,因此

$$\begin{aligned}\max\{d_f(j), d_f(k)\} &= C + a(j) + a(k) + b(k) \\ &\geqslant \max\{d_g(j), d_g(k)\}\end{aligned}$$

从而证明了 $T(f) \geqslant T(g)$,g 也是最优调度.

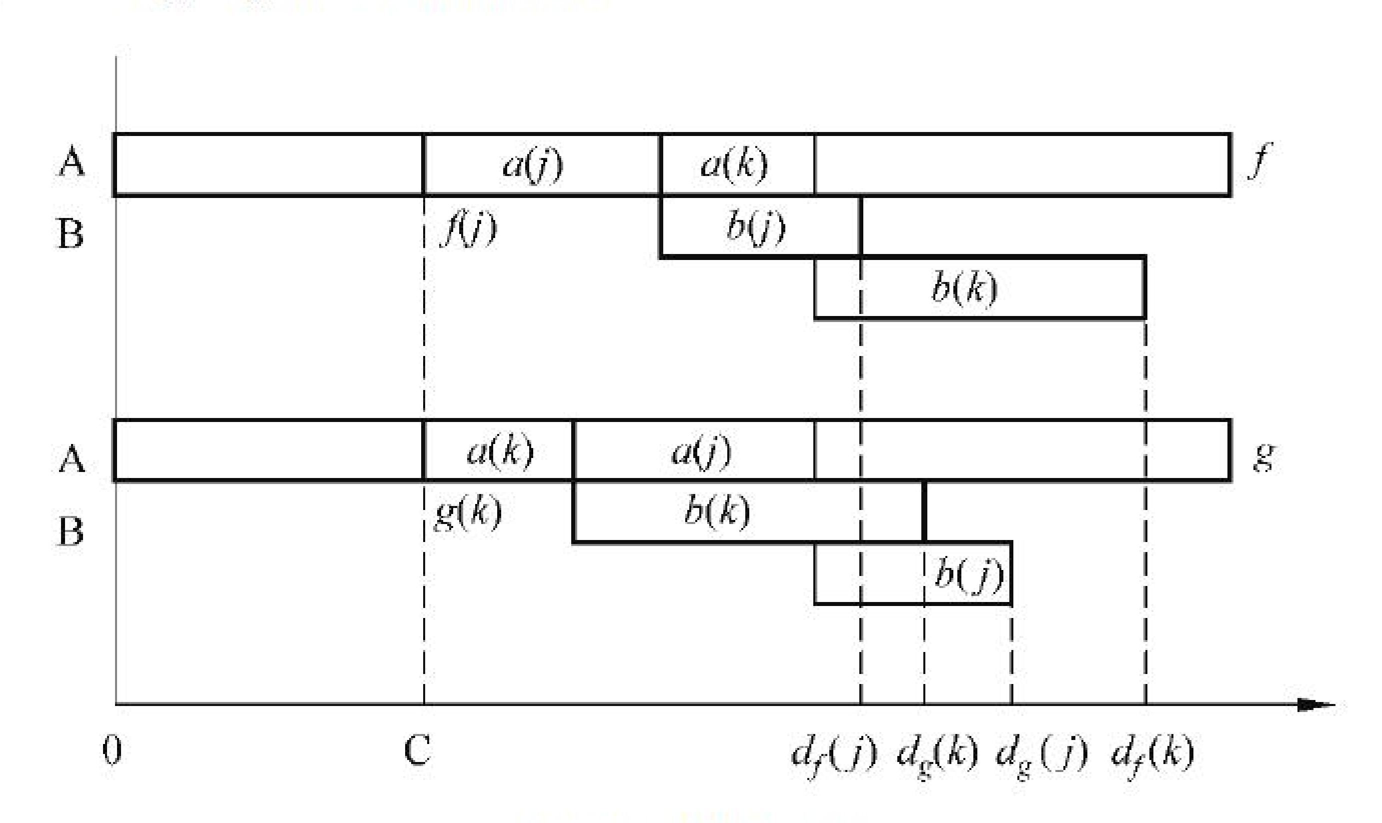


图 4.4　调度 f 和 g

n 个元素的排列中至多含有 $n(n-1)/2$ 个逆序,从一个最优解 f 出发,经过有限次交换,可以消除所有逆序,同时不改变解的最优性,从而证明了用贪心法得到的是最优解.

算法在最坏情况下的时间复杂度是 $O(n\log n)$.

4.16　本题不是组合优化问题,但是可以按照如下的贪心策略进行求解.

贪心策略:从 $a_1, a_2, \cdots$ 顺序检索,找到 b_1 在 A 中第一次出现的位置 a_i,从 a_{i+1} 开始接着找 b_2 在 A 中第一次出现的位置 a_k,从 a_{k+1} 开始找 b_3 第一次在 A 中出现的位置,…,直到找到 b_m 在 A 中第一次出现的位置. 如果上述查找都成功,那么 B 是 A 的子序列. 如果某个 b_j 在 A 中没有找到,那么 B 不是 A 的子序列.

伪码如下:

输入:$A = \langle a_1, a_2, \cdots, a_n \rangle$,$B = \langle b_1, b_2, \cdots, b_m \rangle$

输出:$L[1..m]$ or "No"　　　//$L[j]$表示 b_j 在 A 中位置的下标

```
1.  i←1, j←1
2.  while i≤n and j≤m
3.      if a_i = b_j
4.      then L[j]←i
5.          j←j+1
            if j=m+1 return L
6.      i←i+1
7.  return "No"
```

第 2 行的 while 循环每进入 1 次,n 就加 1,因此至多执行 $O(n)$ 次,循环内部是常数时

间,因此算法的运行时间是 $O(n)$.

考虑下面的命题.

命题 4.14 设算法得到的解是 $L=<l_1,l_2,\cdots,l_m>$,那么对于任意解 $C=<c_1,c_2,\cdots,c_m>$,有 $A[l_j]=A[c_j]=B[j]$且 $l_j\leqslant c_j$, $j=1,2,\cdots,m$.

证 L 和 C 都是解,因此对于 $j=1,2,\cdots,m$ 有 $A[l_j]=A[c_j]=B[j]$. 下面证明对任何 j 有 $l_j\leqslant c_j$.

$j=1$,根据算法描述,$A[l_1]$是在从头开始扫描 A 序列时第一个等于 b_1 的字符,而$A[c_1]$是 A 中某个等于 b_1 的字符,因此 $l_1\leqslant c_1$.

假设对于所有的 k,$l_k\leqslant c_k$. 考虑 $k+1$,因为 $c_k<c_{k+1}$,于是 $l_k<c_{k+1}$,即 $A[c_{k+1}]$排在 $A[l_k]$后面,且 $A[c_{k+1}]=b_{k+1}$. 于是 c_{k+1}是在 l_k 后面等于字符 b_{k+1}的某个位置. 根据算法,l_{k+1}是 l_k 后面第一个等于字符 b_{k+1}的位置,因此 $l_{k+1}\leqslant c_{k+1}$.

设 $L=<l_1,l_2,\cdots,l_m>$是算法得到的解. 如果 B 确实是 A 的子序列,存在解 $C=<c_1,c_2,\cdots,c_m>$,其中 c_j 是 b_j 在 A 中的位置. 如果 $C\neq L$,那么根据命题 4.14 可以依次把 c_1 替换成 l_1,c_2 替换成 l_2,…,c_m 替换成 l_m,最终得到算法的解. 每次替换后得到的仍旧是子序列. 于是,如果问题的实例有解,算法的解就是正确的.

需要说明的是:本题不是使得目标函数达到最大或最小的组合最优化问题,不同的解之间没有哪个解更优的区别,但是在算法设计中同样体现了贪心选择的思想:把 B 中字符与 A 匹配时,在满足子序列要求的前提下尽可能选择最前面的字符.

4.17 (1) 先证明有关最小生成树的一个性质.

命题 4.15 设 T 是图 G 的一棵最小生成树,C 是 G 中的一个圈,如果 C 中边的权彼此不等,那么在 C 中权最大的边不属于 T.

证 设 e 是在 C 中权最大的边. 假若 e 在 T 中,取 C 中不在 T 中的边 e',由于 $w(e)$与 $w(e')$彼此不等,于是 $w(e')<w(e)$. 用 e'替换 e,得到的是一棵 G 的生成树 T^*,且 $w(T^*)<w(T)$,与 T 的最小性矛盾.

$\forall e\in E$,令 $s(e)=-w(e)$是边 e 的权值,构造带权图 $G'=<V,E,S>$. 可以证明 G'的最小生成树 T'满足本题所要求的最佳带宽性质.

命题 4.16 $\forall u,v\in V$,P'是在上述生成树 T' 中连接 u 与 v 的唯一路径,那么 $w(P')=w(u,v)$.

证 用反证法. 假若不是,那么 G' 中存在结点 u 和 v,且有一条 u-v 路径 P,使得 $w(P')<w(P)$. 设 $e'=(x,y)$是 T'中带宽最小的边,即 $\forall e\in P'$,$w(P')=w(e')<w(e)$. 又由 $w(P')<w(P)$,P 上任何边的带宽大于 $w(e')$,因此 e'不在路径 P 上. 于是 G'中存在一个圈 C,如图 4.5 所示,C 从 x 经过 P'路径上的一段到 u,然后经过路径 P 到 v,再经过路径P'上的一段到 y,再经过边 e'最后回到 x. 从上面的分析可知,e'是 C 上带宽最小的边,即 $w(e')<w(e)$. 而在 T'中边 e 的权 $s(e)=-w(e)$,即 $s(e')>s(e)$,即 e'是圈 C 中权最大的边. 根据命题 4.15,边权最大的边不属于任何最小生成树,与 T'是 G'的最小生成树矛盾. 由此可知不存在具有更佳带宽的 u-v 路径 P. 从而有 $w(P')=w(u,v)$.

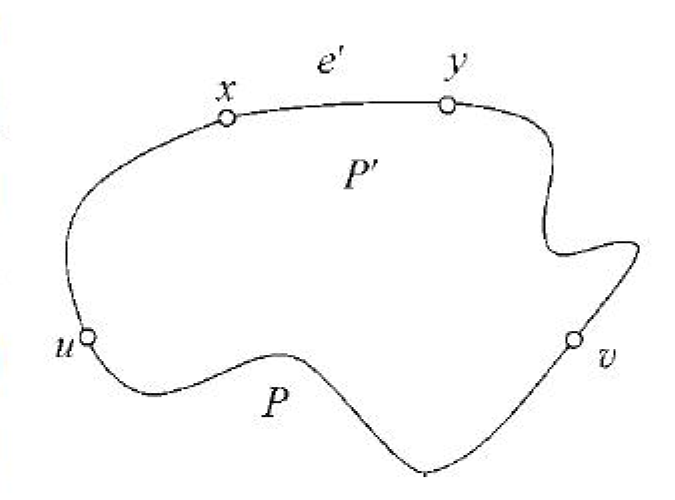


图 4.5 G'中的圈 C

(2) 根据上述命题,可以设计算法如下:

1. 对所有的 $e \in E$,令 $s(e)=-w(e)$.

2. 求带权图 $G'=<V,E,S>$ 的一棵最小生成树.

根据 Kruskal 算法,最坏情况下的时间复杂度是 $T(n)=O(m\log m)=O(m\log n)$.

4.18 (1) 按照标号从小到大考查广告. A 是被选取的广告标号集合,初始 $A=\{1\}$. 设 j 是当前 A 中最大标号. 考查第 k 个项目时,需检验 $s(k)\geqslant d(j)$. 若是,则把 k 加入 A;否则考查 $k+1$ 项.

命题 4.17 对任意正整数 k,算法选择了 k 项广告 $i_1=1,i_2,\cdots,i_k$,存在包含 $i_1=1,i_2,\cdots,i_k$ 的最优解.

证 对 k 归纳.

$k=1$,一定存在最优解包含 1. 如若不然,设最优解 A'中广告的最小标号是 j,将 j 替换成 1,即

$$A^* = (A' - \{j\}) \cup \{1\}$$

那么 A^* 是包含标号 1 的最优解.

假设对任意正整数 k 命题为真,即算法到第 k 步选择了 k 项广告 $i_1=1,i_2,\cdots,i_k$,且存在包含 $i_1=1,i_2,\cdots,i_k$ 的最优解 A,那么有

$$A = \{i_1,i_2,\cdots,i_k\} \cup B$$

令 S'是 $S-\{i_1,i_2,\cdots,i_k\}$中开始时间不小于 $d(k)$(与已经选中的广告相容)的那些广告的集合. 那么不难证明 B 是 S'的一个最优解. 根据归纳基础的证明可以知道,第 1 步选择标号最小的广告总能导致最优解,因此 S'中有一个包含广告 i_{k+1} 的最优解 B^*. B^* 与 B 都是最优解,因此 $|B^*|=|B|$. 在 A 中用 B^* 替换 B 就得到如下最优解

$$A' = \{i_1,i_2,\cdots,i_k\} \cup B^* = \{i_1,i_2,\cdots,i_k,i_{k+1},\cdots\}$$

这恰好包含了算法前 $k+1$ 步的选择. 根据归纳法,命题得证.

算法在最坏情况下的时间复杂度为 $O(n)$.

(2) 用动态规划算法,设 $F[k]$表示考虑前 k 个广告且选中第 k 个广告的最大效益. 定义 $p(k)$是与 k 相容(时间不重叠)且标号小于 k 的广告中的最大标号. 那么 $F[k]$满足如下递推关系:

$$F[k] = \max\{F[k-1],F[p(k)]+v(k)\}, \quad k>1$$
$$p(k) = \max\{i \mid i \in \{1,2,\cdots,k-1\} \text{ 且 } d(i) \leqslant s(k)\}$$
$$F[1] = v(1)$$

用标记函数 $i[k]$记下当选到第 k 项广告时,紧接在它前边安排的广告标号,即

$$i[k] = \begin{cases} p(k), & \exists\, p(k) \in \{1,2,\cdots,k-1\} \text{ 且 } F[p(k)]+v(k) \geqslant F[k-1] \\ 0, & \text{否则} \end{cases}, \quad k \geqslant 1$$

算法得到的最大效益是

$$\max\{F[k] \mid k = 1,2,\cdots,n\}$$

算法输出数组 F 和 i 以及取得最大效益时的 k 值(最后一项广告的标号).

为追踪解 A 需要考查标记函数 i. 具体方法是:将 k 加入解 A. 接着追踪 $i[k]$,如果 $i[k]=j$,那么 j 是在 k 前面被选中的广告标号. 把 j 加入 A,将 k 更新为 j,继续追踪新的 $i[k]$,直到 $i[k]=0$ 为止.

上述算法在最坏情况下的时间复杂度是 $O(n^2)$.

4.19 采用贪心法.

贪心策略：按照单位存储空间被检索的概率 f_i/s_i 从大到小排序文件使得

$$\frac{f_1}{s_1} \geqslant \frac{f_2}{s_2} \geqslant \cdots \geqslant \frac{f_n}{s_n}$$

然后按照文件排序的顺序依次存入磁带.

命题 4.18 按上述次序存储磁盘的平均检索时间达到最小.

证 设某个最优解 OPT(I)的磁盘文件顺序是 $i_1, i_2, \cdots, i_n$，则平均检索时间是

$$t(\mathrm{OPT}(I)) = f_{i_1}s_{i_1} + f_{i_2}(s_{i_1} + s_{i_2}) + f_{i_3}(s_{i_1} + s_{i_2} + s_{i_3}) + \cdots + f_{i_n}(s_{i_1} + \cdots + s_{i_n})$$

如果 OPT(I)与算法的解不等，那么它们的区别在于 OPT(I)中存在逆序. 若在 OPT(I)中存在逆序，一定存在相邻的逆序，比如第 i 个文件和第 j 个文件构成相邻的逆序. 交换文件 i 和 j 得到解 $P(I)$，那么

$$\begin{aligned} t(\mathrm{OPT}(I)) - t(P(I)) &= [f_i s_i + f_j(s_i + s_j)] - [f_j s_j + f_i(s_j + s_i)] \\ &= f_j s_i - f_i s_j \geqslant 0 \quad \left(\frac{f_i}{s_i} \leqslant \frac{f_j}{s_j}\right) \end{aligned}$$

从而证明了从 OPT(I)出发，交换具有相邻逆序的元素后仍旧得到最优解. 每进行 1 次交换，消除 1 个逆序. 至多经过 $n(n-1)/2$ 次交换，就消除了所有的逆序，从而得到算法的解，因此算法的解也是最优解.

算法最坏情况下时间复杂度是 $O(n\log n)$.

4.20 使用贪心法.

贪心策略：排序 r_i 为递减次序，使得 $r_1 \geqslant r_2 \geqslant \cdots \geqslant r_n$，依次购买.

命题 4.19 按照上述顺序购买软件许可证花费的总钱数最少.

证 设最优解为 OPT(I)，假设在 OPT(I)存在逆序，即 $r_j < r_i$，但是 j 在 i 前面购买. 一定有相邻的逆序，即存在 i 和 j，使得 j 在 i 前面与 i 相邻. 设 j 是第 t 个月购买，i 是第 $t+1$ 个月购买. 交换 i 与 j 得到解 $S(I)$，那么花费之差

$$\begin{aligned} & V(\mathrm{OPT}(I))V(S(I)) \\ &= (r_i^{t+1} \times 1000 + r_j^t \times 1000) - (r_i^t \times 1000 + r_j^{t+1} \times 1000) \\ &= [r_i^t(r_i - 1) - r_j^t(r_j - 1)] \times 1000 \end{aligned}$$

由于 $r_i \geqslant r_j$，于是 $r_i - 1 \geqslant r_j - 1$ 且 $r_i^t \geqslant r_j^t$，得到上式$\geqslant 0$.

算法的时间复杂度为 $O(n\log n)$.

4.21 使用贪心法. 类似于习题 4.9. 把每个集合看成一个进程，进程 A_i 的开始时间为 a_i，结束时间为 b_i. S 相当于测试点的集合，求最小的 S，使得每个进程至少包含 S 中一个测试点.

贪心策略：对 b_i 按照从小到大的次序排序，将 b_1 放入 S. 将 b_1 记作当前测试点 x. 按照顺序依次检查 $A_2, A_3, \cdots, A_n$. 如果 $a_i \leqslant x$，则 S 中已包含 A_i中的某个整数，接下去检查 A_{i+1}. 如果 $a_i > x$，那么把 b_i加入 S.，并将 x 更新为 b_i. 伪码如下：

1. 按照 b_i从小到大对集合 $A_1, A_2, \cdots A_n$排序，使得 $b_1 \leqslant b_2 \leqslant \cdots \leqslant b_n$.

2. 令 $S \leftarrow \{b_1\}$

3. for $i \leftarrow 2$ to n do

```
4.        if x<a_i
5.        then S←S∪{b_i}
6.            x←b_i
7. return S
```

命题 4.20 按照上述算法选出的 S 是与每个 A_i 相交的最小集合.

证 对步数 k 归纳.

$k=1$,存在最优解 S 含有 b_1. 假设 T 是最优解,T 含有 A_1 中的 x_i. 如果 x_i 不是 b_1,用 b_1 替换 x_i 得到 T',由于 $x_i<b_1$,每个集合是连续整数,含有 x_i 的集合一定含有 b_1,于是 T' 也是最优解.

假设对于任意 k,算法 k 步选择都导致最优解,即存在最优解

$$T = T_k \cup B$$

其中 $T_k=\{b_1,b_{i2},\cdots,b_{ik}\}$ 是算法前 k 步选择的数的集合. 如果 T_k 与 $A_{k+1},\cdots,A_n$ 不交的集合为 $A_{j1},A_{j2},\cdots,A_{jl}$,那么 B 是 $A_{j1},A_{j2},\cdots,A_{jl}$ 的最优解. 若不然,存在更好的解 B^*,那么 $T_k\cup B^*$ 将是比 T 更好的解,与归纳假设矛盾.

根据归纳基础,$A_{j1},A_{j2},\cdots,A_{jl}$ 有一个最优解 B' 含有算法第 $k+1$ 步选择的元素 $b_{i(k+1)}$,且 $|B'|=|B|$,于是

$$T' = T_k \cup B' = T_k \cup \{b_{i(k+1)},\cdots\} = \{b_1,b_{i2},\cdots,b_{ik},b_{i(k+1)},\cdots\}$$

且 $|T'|=|T|$,于是算法到第 $k+1$ 步的选择也将导致最优解. 根据归纳法命题得证.

算法的时间复杂度为 $O(n\log n)$.

第 5 章

回溯与分支限界

5.1 内容提要

1. 基本概念

解空间 搜索问题的解所在的集合，又称为搜索空间. 解空间通常可以安排成树形结构，常用解空间有子集树、排列树等.

回溯算法 遵照某种策略搜索解空间从而找出解的过程. 常用的搜索策略有：深度优先、宽度优先、规则优先等.

分支限界算法 回溯算法的一种特例. 在回溯算法运行过程中，为加快算法的速度，尽可能多地在解空间中进行剪枝，设立新的约束条件——代价函数和界. 当代价函数值小于(对于最大化问题)界的时候即进行剪枝，这种回溯算法称为分支限界算法.

代价函数 以搜索树的任何的结点 v 开始搜索以 v 为根的子树，在这个范围可能得到可行解，代价函数 $f(v)$ 表示在该子树中可行解的目标函数值的一个上界(对于最大化问题).

界函数 函数在搜索树中某结点的函数值是算法搜索到该结点时，已经得到的可行解的目标函数的最大值.

多米诺性质：$P(x_1, x_2, \cdots, x_{i+1})$为真蕴含 $P(x_1, x_2, \cdots, x_i)$为真.

2. 算法

***n* 皇后问题的回溯算法** 该问题是：在 $n \times n$ 的棋盘上放置 n 个皇后，使得任何两个皇后不能相互攻击，即棋盘的同一行、同一列、主对角线的平行线、副对角线的平行线上都不能有两个以上的皇后. 给出所有的放置方法. 该问题的回溯算法的解空间是一个树高为 n 的 n 叉完全正则树，树中第 k 层的边表示棋盘中第 k 行的皇后位置，搜索策略是深度优先，在每个结点处检查是否有皇后冲突，若有，则回溯；若无，则向下分支. 若以检查两皇后是否冲突作为基本运算，该算法的最坏情形复杂性为 $O(3n \times 2n^n) = O(n^{n+1})$.

0-1 背包问题的回溯算法 0-1 背包问题的描述见第 3 章. 该问题的回溯算法的解空间是一个树高为 n 的 2 叉完全正则树，即子集树，树的第 k 层中，关联每个顶点的两条边分别表示第 k 个物品放入或不放入背包中(分别标记为 1 和 0)，搜索策略是深度优先，在每个结点处检查放入背包物品重量之和是否大于背包的承载量，若是，则回溯；否则向下分支. 若以数的大小比较作为基本运算，则该算法的最坏情形复杂性为 $O(2^n)$.

货郎问题(TSP)的回溯算法　货郎问题是指,某售货员要到若干城市去推销商品,各城市之间的距离为已知;他要选定一条从驻地出发经过所有城市最后回到驻地的一条周游路线,使得总的路程最短. 其数学模型是:已知一个带权图完全图(结点代表城市,边代表城市之间的道路,权代表城市之间的距离,如两个城市之间无直接连接的道路,设其权为∞,所以权为正数或无穷),求权和最小的一条哈密顿回路. 该问题的回溯算法的解空间是一个树高为 n 的排列树,树中关联根结点的第 1 层的边只有一条,表示该售货员所在的城市编号,第 2 层有 $n-1$ 条边(n 为城市的个数),分别表示下一步要到达的城市编号,第 $k(k\geqslant 2)$层中每个顶点有 $n-(k-1)$条边,分别表示下一步要到达的城市编号. 搜索策略是深度优先,在每个结点处计算所经过的路径的长度. 若以数的加法作为基本运算,该算法的最坏情形复杂性为 $O((n-1)!)$.

装载问题的回溯算法　装载问题是指,设有重量分别为 w_i 的 n 个集装箱,将其装上 2 艘载重分别为 c_1 和 c_2 的轮船,已知 $\sum_{i=1}^{n} w_i \leqslant c_1 + c_2$,问是否存在一种合理的装载方案将 n 个集装箱装上轮船? 该问题的回溯算法的解空间是一个子集树,搜索策略是深度优先,在每个结点处检查装入的集装箱重量之和是否超过第一艘船的承载量. 遍历整个解空间后得到使得装入第一艘船的集装箱的重量最大. 最后检查剩下集装箱的重量之和是否小于第二条船的承载量,如是则回答"是";否则回答"否". 若以数的加法作为基本运算,该算法的最坏情形复杂性为 $O(2^n)$.

图的 *m* 着色问题　该问题是指,给定无向连通图 G 和 m 种颜色,用这些颜色给图的顶点着色,每个顶点一种颜色. 如果要求 G 的每条边的两个顶点是不同颜色,给出所有可能的着色方案;如果不存在着这样的方案,则回答"No". 该问题的回溯算法的解空间是一个 m 叉完全正则树,搜索策略还是深度优先. 若以颜色比较作为基本运算,该算法在最坏情形下的复杂性为$O(nm^n)$.

背包问题的分支限界算法　问题的描述见主教材第 3 章. 该问题的分支限界算法是在回溯算法的基础上,定义如下代价函数:结点$<x_1, x_2, \cdots, x_k>$的代价函数值为:在$<x_1, x_2, \cdots, x_k, x_{k+1}, \cdots, x_n>$中,无论 $x_{k+1}, \cdots, x_n$ 取何值,$\sum_{i=1}^{n} v_i x_i$ 的一个上界;界函数在结点$<x_1, x_2, \cdots, x_k>$处的函数值是在此结点之前已经找到的方案中,放入背包物品的最大总价值. 若以数的加法作为基本运算,该算法在最坏情形下的复杂性依然为 $O(2^n)$.

最大团问题的分支限界算法　该问题为求无向图的顶点个数最多的团. 该问题的分支限界算法中的解空间是一颗子集树,代价函数是 $F=c_n+n-k$,其中:c_n 为目前形成的团的顶点数(初始为 0),k 为目前检索的子集树的结点的层数,即已经检索过的顶点数. 以检查两个结点是否相邻作为基本运算,该算法在最坏情形下的复杂性为 $O(n2^n)$.

货郎问题(TSP)的分支限界算法　在回溯算法的基础上,定义如下代价函数:$L=\sum_{j=1}^{k} c_j + \sum_{i_j \notin B} l_{i_j} + l_{i_k}$,其中$\sum_{j=1}^{k} c_j$为已选定巡回路线的长度,$\sum_{i_j \notin B} l_{i_j} + l_{i_k}$ 为经过剩余结点回到结点 1 的最短距离的一个下界,界函数值为当前得到的最短巡回路线长度. 若以数的加法作为基本运算,该算法在最坏情形下的复杂性依然为 $O((n-1)!)$.

圆排列问题的分支限界算法 该问题是指,给定 n 个圆,已知每个圆的半径 r_i. 现将它们放到矩形框中,各圆与矩形底边相切,求具有最小长度 l_n 的圆排列. 该问题的分支限界算法中的解空间是一个排列树,搜索策略是深度优先,代价函数 L_k 为 $x_k+(2n-2k+1)r+r_1$,其中:r 为后面待选的 $n-k$ 个圆以及第 k 个圆中最小半径的值,x_k 表示第 k 个圆的圆心的横坐标. 界函数在 $<i_1,i_2,\cdots,i_k>$ 处的值是当前已得到的最小圆排列长度. 若以数的加法作为基本运算,该算法在最坏情形下的复杂性依然为 $O((n+1)!)$.

连续邮资问题的回溯算法 该问题是指,设有 n 种不同面值的邮票,每个信封至多贴 m 张邮票,试给出邮票面值的最佳设计(面值为正整数值),使得到从 1 开始,增量为 1 的连续邮资区间最大. 该问题的回溯算法中的解空间并不是算法一开始就能确定下来,需要边搜索边构造. 设在结点 $<x_1,x_2,\cdots,x_i>$ 处,邮资最大连续区间为 $\{1,2,\cdots,r_i\}$,则 x_{i+1} 的取值范围为 $\{x_i+1,\cdots,r_i+1\}$.

5.2 习　　题

要求:对于回溯法,要说明解向量、搜索树结构、搜索策略、代价函数(如果存在)等,并给出最坏情况下的时间复杂度函数. 对于给定的数的输入实例,求出所有的可行解(或一个最优解).

5.1 用回溯法求下列不等式的所有的整数解. 要求给出伪码和解.

$$\begin{cases}3x_1+4x_2+2x_3\leqslant 12\\ x_1,x_2,x_3\ \text{为非负整数}\end{cases}$$

5.2 最小重量机器设计问题. 某设备需要 4 种配件,每种 1 件. 有 3 个供应商提供这些配件,表 5.1 给出相关的价格和每种配件的重量. 从中选择这 4 种配件,使得总价值不超过 120,总重量最轻.

表 5.1 产品和供应商信息表

零件编号	供应商 1		供应商 2		供应商 3	
	价格	重量	价格	重量	价格	重量
1	10	5	8	6	12	4
2	20	8	21	10	30	5
3	40	5	42	4	30	10
4	30	20	60	10	45	15

5.3 如图 5.1 所示,一个 4 阶 Latin 方是一个 4×4 的方格,在它的每个方格内填入 1、2、3 或 4,并使得每个数字在每行、每列都恰好出现一次. 用回溯法求出所有第一行为 1、2、3、4 的 4 阶 Latin 方. 将每个解的第 2 行至第 4 行的数字从左到右写成一个序列. 例如图 5.1 中的 Latin 方对应于解:$<3,4,1,2,4,3,2,1,2,1,4,3>$. 给出所有可能的 4 阶 Latin 方.

1	2	3	4
3	4	1	2
4	3	2	1
2	1	4	3

图 5.1 Latin 方

5.4 应用回溯算法给出{1,2,3,4}的所有置换.

5.5 给出 8 皇后问题的一个广度优先回溯算法,并分析该算法的时

间复杂度.

5.6 子集和问题. 设 n 个不同的正数构成集合 S,求出使得和为某数 M 的 S 的所有子集.

5.7 分派问题. 给 n 个人分配 n 件工作,给第 i 个人分配第 j 件工作的成本是 $C(i,j)$. 试求成本最小的工作分配方案.

5.8 电路板排列问题. 设 $B=\{1,2,\cdots,n\}$ 是 n 块电路板的集合,$L=\{N_1,N_2,\cdots,N_m\}$ 是 m 块连接块的集合. 对 $j=1,2,\cdots,m$,连接块 $N_j\subseteq B$ 相当于一条导线,把属于 N_j 的所有电路板连起来. 如图 5.2 所示,$B=\{1,2,\cdots,8\}$,其中位置 1,2,…,8 是插槽,每个插槽可以插入一块电路板. 当电路板按照某种排列顺序全部插入后,一些连接块的导线有可能从一块电路板插槽跨到相邻的另一个插槽. 比如图 5.2 中的 5 块连接块是:

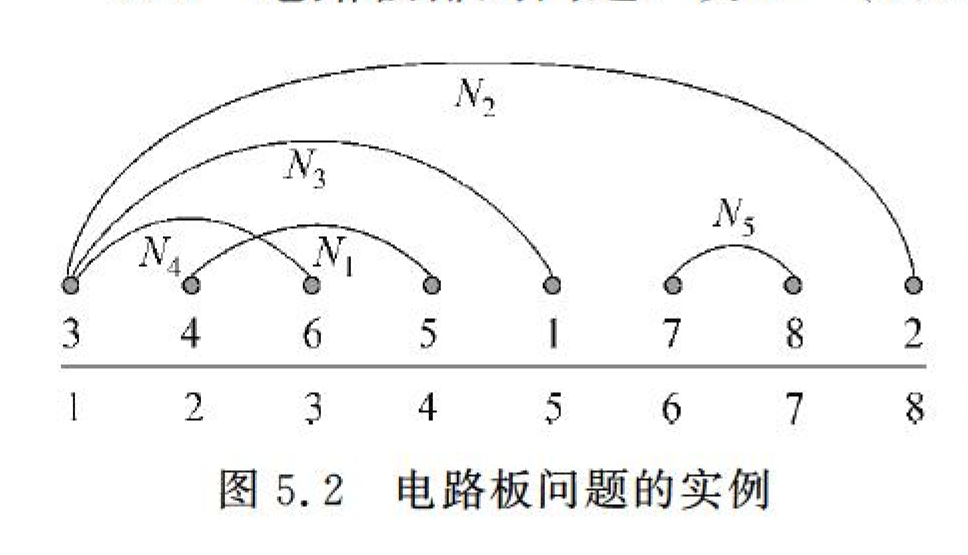


图 5.2 电路板问题的实例

$$N_1=\{4,5,6\},\quad N_2=\{2,3\},\quad N_3=\{1,3\},\quad N_4=\{3,6\},\quad N_5=\{7,8\}$$

如果电路板按照 3,4,6,5,1,7,8,2 的顺序排列,那么横跨插槽 1 和 2 的连线数是 3,即连接块 N_2、N_3 和 N_4;横跨插槽 2 和 3 的连线数是 4,即连接块 N_1、N_2、N_3 和 N_4;…;横跨插槽 7 和 8 的连线数是 1,即连接块 N_2;其中最大连线数是 4. 称 4 是排列 $X=<3,4,6,5,1,7,8,2>$ 的排列密度,记作 density(X). 对于电路板的不同排列 X 和 X',其排列密度值可能是不一样的. 电路板问题是指,给定集合 B 和 L,求使得排列密度最小的电路板排列.

(1) 对上述电路板问题的实例,求出该实例的最优解.

(2) 不遍历搜索树,用数学方法证明你的解是最优的.

5.9 设有 n 项任务由 k 个可并行操作的机器完成,完成任务 i 所需要的时间是 t_i,求一个最佳任务分配方案,使得完成时间(即从时刻 0 计时,到最后一台机器停止的时间)达到最短.

5.10 在 5.9 题中,假设每个任务有个完成期限 B_i,以及超出期限的罚款数 f_i. 试求一个最佳任务分配方案,使得完成所有任务的总罚款最少.

5.11 哨兵布置问题. 一个博物馆由排成 $m\times n$ 个矩形阵列的陈列室组成,需要在陈列室中设立哨位,每个哨位上的哨兵除了可以监视自己所在陈列室外,还可以监视他上、下、左、右四个陈列室,试给出一个最佳哨位安排方法,使得所有陈列室都在监视之下,但使用的哨兵最少.

5.3 习题解答与分析

5.1 34 个解. 即

(0,0,0), (0,0,1), (0,0,2), (0,0,3), (0,0,4), (0,0,5), (0,0,6),
(0,1,0), (0,1,1), (0,1,2), (0,1,3), (0,1,4), (0,2,0), (0,2,1),
(0,2,2), (0,3,0), (1,0,0), (1,0,1), (1,0,2), (1,0,3), (1,0,4),
(1,1,0), (1,1,1), (1,1,2), (1,2,0), (2,0,0), (2,0,1), (2,0,2),
(2,0,3), (2,1,0), (2,1,1), (3,0,0), (3,0,1), (4,0,0)

5.2 按照价格从小到大对零件排序. 设解向量为$<x_1,x_2,\cdots,x_n>$,$x_i=j$ 表示第 i 号零件由 j 号供应商供货. $1\leqslant x_j\leqslant m$. 结点$<x_1,x_2,\cdots,x_k>$表示已经选择了前 k 号零件的供应商,正在处理第 $k+1$ 号零件.

约束条件:选择了下一个零件后总价格不超过 120.

代价函数:

$$\sum_{i=1}^{k} w_{ix_i} + \sum_{j=k+1}^{n} \min_{l=1,2,\cdots,m} \{w_{jl}\}$$

其中 w_{jl} 表示第 l 个供应商 j 号零件的重量.

解:对实例<3,1,2,3>,总重量为 31,价值为 119.

5.3 有 24 个 Latin 方,如图 5.3 所示.

1	2	3	4
2	1	4	3
3	4	1	2
4	3	2	1

1	2	3	4
2	1	4	3
3	4	2	1
4	3	1	2

1	2	3	4
2	1	4	3
4	3	1	2
3	4	2	1

1	2	3	4
2	1	4	3
4	3	2	1
3	4	1	2

1	2	3	4
2	3	4	1
3	4	1	2
4	1	2	3

1	2	3	4
2	3	4	1
4	1	2	3
3	4	1	2

1	2	3	4
2	4	1	3
3	1	4	2
4	3	2	1

1	2	3	4
2	4	1	3
4	3	2	1
3	1	4	2

1	2	3	4
3	1	4	2
2	4	1	3
4	3	2	1

1	2	3	4
3	1	4	2
4	3	2	1
2	4	1	3

1	2	3	4
3	4	1	2
2	1	4	3
4	3	2	1

1	2	3	4
3	4	1	2
2	3	4	1
4	1	2	3

1	2	3	4
3	4	1	2
4	1	2	3
2	3	4	1

1	2	3	4
3	4	1	2
4	3	2	1
2	1	4	3

1	2	3	4
3	4	2	1
2	1	4	3
4	3	1	2

1	2	3	4
3	4	2	1
4	3	1	2
2	1	4	3

1	2	3	4
4	1	2	3
2	3	4	1
3	4	1	2

1	2	3	4
4	1	2	3
3	4	1	2
2	3	4	1

1	2	3	4
4	3	1	2
2	1	4	3
3	4	2	1

1	2	3	4
4	3	1	2
3	4	2	1
2	1	4	3

1	2	3	4
4	3	2	1
2	1	4	3
3	4	1	2

1	2	3	4
4	3	2	1
2	4	1	3
3	1	4	2

1	2	3	4
4	3	2	1
3	1	4	2
2	4	1	3

1	2	3	4
4	3	2	1
3	4	1	2
2	1	4	3

图 5.3 24 个 Latin 方

5.4 解向量为$<x_1,x_2,x_3,x_4>$,搜索空间是排列树. 有 24 个置换.

5.5 解向量为$<x_1,x_2,\cdots,x_8>$,搜索空间是 8 叉树. 在代表部分向量$<x_1,x_2,\cdots,x_k>$的结点处,下一步分支条件是 x_{k+1} 与 $x_1,x_2,\cdots,x_k$ 相容(不在同一行、同一列,也不在同一条斜线上). 搜索是按广度优先顺序遍历这棵树. 对于 n 后问题,最坏情况下的时间复杂度为 $O(n^n)$.

5.6 设 $S=\{a_1,a_2,\cdots,a_n\}$. 求 S 满足条件 $\sum_{a_i\in A} a_i = M$ 的所有的子集 A. 用回溯算法. 解向量为$<x_1,x_2,\cdots,x_n>$,$x_i=0,1$. 其中 $x_i=1$ 当且仅当 $a_i\in A$. 搜索空间为子集树. 部分向量$<x_1,x_2,\cdots,x_k>$表示已经考虑了对 $a_1,a_2,\cdots,a_k$ 的选择. 结点分支的约束条件为

$$B(i) = \sum_{i=1}^{k} a_i x_i < M \quad 且 \quad a_{k+1} \in S - \{a_1,a_2,\cdots,a_k\}$$

最坏情况下算法的时间复杂度为 $O(2^n)$.

5.7 设 n 个人的集合是 $\{1,2,\cdots,n\}$, n 项工作的集合是 $\{1,2,\cdots,n\}$, 每个人恰好 1 项工作.

$$把工作 j 分配给 i \Leftrightarrow x_i = j, \quad i,j = 1,2,\cdots,n$$

解向量是 $X=<x_1,x_2,\cdots,x_n>$, 分配成本是 $C(X) = \sum_{i=1}^{n} C(i,x_i)$. 搜索空间是排列树. 部分向量 $<x_1,x_2,\cdots,x_k>$ 表示已经考虑了对人 $1,2,\cdots,k$ 的工作分配. 结点分支的约束条件为:

$$x_{k+1} \in \{1,2,\cdots,n\} - \{x_1,x_2,\cdots,x_k\}$$

可以设立代价函数:

$$F(x_1,x_2,\cdots,x_k)$$
$$= \sum_{i=1}^{k} C(i,x_i) + \sum_{i=k+1}^{n} \min\{C(i,t) \mid t \in \{1,2,\cdots,n\} - \{x_1,x_2,\cdots,x_k\}\}$$

界 B 是已得到的最好可行解的分配成本. 如果代价函数大于界, 则回溯.

算法最坏情况下的时间复杂度是 $O(nn!)$.

5.8 搜索空间为排列树. 叶结点 $<x_1,x_2,\cdots,x_n>$ 对应排列 X. 在第 i 层, 结点 $<x_1,x_2,\cdots,x_i>$ 表示部分排列. 搜索策略为深度优先. 在结点 $<x_1,x_2,\cdots,x_i>$, 令 $B=\{x_1,x_2,\cdots,x_i\}$, 下一步选择 x_{i+1}. 其约束条件是

$$x_{i+1} \in \{1,2,\cdots,n\} - B$$

界为目前得到的最小排列密度.

在电路板 x_i 和 x_{i+1} 之间, 如何计算横跨插槽 x_i 和 x_{i+1} 的连线数? 令 $total[j]$ 是连接块 j 所连接的电路板总数, $now[j]$ 是 $x_1,x_2,\cdots,x_i$ 中已经包含在 N_j 中的电路板数, 那么

$$N_j 的连线跨越位置 i 和 i+1 的电路板 \Leftrightarrow now[j] < total[j] 且 now[j] > 0$$

即 N_j 的部分连线连接 $x_1,x_2,\cdots,x_i$ 中的电路板(因为 $now[j]>0$); 另一部分连线连接除了 $x_1,x_2,\cdots,x_i$ 之外的其他电路板(因为 $now[j]<total[j]$). 令

d_i: 从位置为 1 到 i 的电路板的最大排列密度,

$$S_{i+1} = \{j \mid now[j] < total[j], now[j] > 0, j = 1,2,\cdots,m\}$$

针对本题给出的实例, $N_1=\{4,5,6\}$, $N_2=\{2,3\}$, $N_3=\{1,3\}$, $N_4=\{3,6\}$, $N_5=\{7,8\}$. 如果排列是 3,4,6,5,1,7,8,2, 当 $i=2$ 时, 有

$$now[1] = now[2] = now[3] = now[4] = 1, \quad now[5] = 0$$
$$total[1] = 3$$
$$total[2] = total[3] = total[4] = total[5] = 2$$

因此 N_1,N_2,N_3,N_4 的连线跨越位置为 2 和 3 的电路板. 在分支点 $<x_1,x_2,\cdots,x_i>$ 选择 x_{i+1} 后的估计为:

$$d_{i+1} = \max\{d_i, |S_{i+1}|\}$$
$$\text{density}(X) = d_n$$

算法在最坏情况下的时间复杂度为 $O(mn!)$.

5.9 设 n 个任务的标号分别为 $1, 2,\cdots, n$, k 个处理器的标号分别为 $1, 2,\cdots, k$.

算法主要步骤如下:

1. 给出将 n 个任务分到 k 个处理器上的分配方法.

2. 在每个分配方法下,算出每个处理器上所分配到的任务的执行时间,并求这些执行时间的最大值,该最大值即为这个分配方法的执行时间.

3. 求出所有分配方法的执行时间的最小值.

步骤 1 可以用深度优先策略遍历如图 5.4 所示的完全 k 元正则树实现.

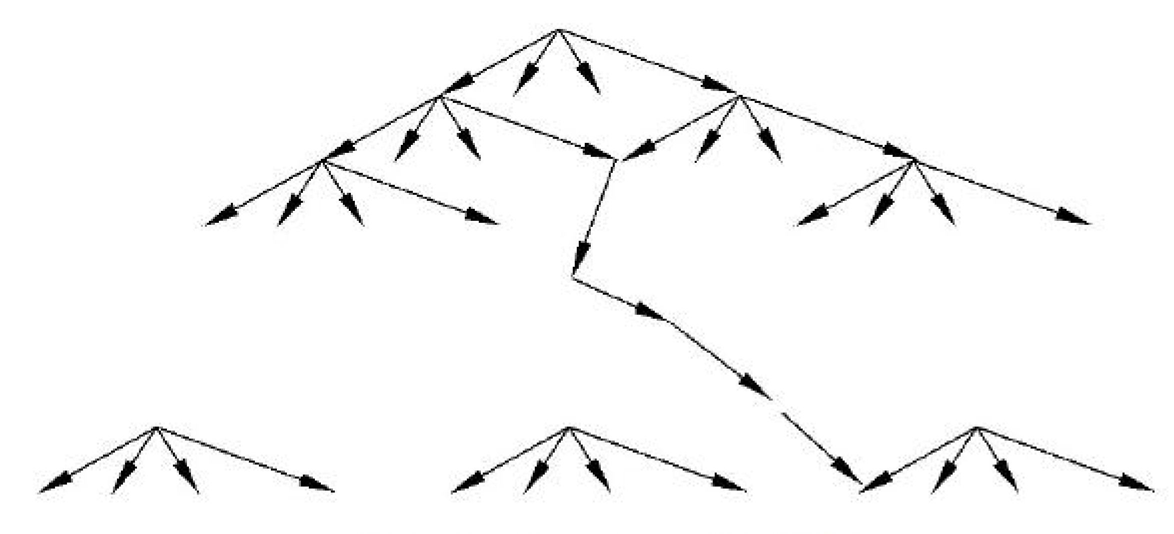


图 5.4 k 元完全正则树

树的根结点分支出来的 k 个条边分别标记 k 个处理器的标号,表示任务 1 所分配的处理器标号;树的第一层每个结点所分支出的 k 条边也分别标记 k 个处理器的标号,表示任务 2 所分配的处理器标号,以此类推.

从根到叶结点路径上的边的标记序列 $<i_1, i_2, \cdots, i_n>$(其中 $1 \leqslant i_j \leqslant k, 1 \leqslant j \leqslant n$)表示了 n 个任务在 k 个处理器上的如下分配方案:任务 $1, 2, \cdots, n$ 分别分配到标号为 $i_1, i_2, \cdots, i_n$ 的处理器上执行.

按深度优先方法遍历这棵树即可完成了第一步的工作,得到 k^n 个分配方案.

步骤 2 可以按如下算法完成:

在分配方案 $<i_1, i_2, \cdots, i_n>$ 下,如 $i_k = i_l$,则表明任务 k、l 被分配到同一个处理器 i_k 上执行. 遍历序列 $i_1, i_2, \cdots, i_n$,找到每个处理器上分配到的所有任务 $w_1, w_2, \cdots, w_u$,则在这个分配方案下,该处理器的执行时间为 $t_{w_1} + t_{w_2} + \cdots + t_{w_u}$. 再在所有处理器的执行时间中求出最大值,则该最大值即为分配方案 $<i_1, i_2, \cdots, i_n>$ 的执行时间. 该步骤的执行时间为 $O(n k^n)$.

步骤 3 是在 k^n 个分配方案的执行时间中求最小值,执行时间为 $O(k^n)$. 至此算法结束. 总的执行时间为 $O(n k^n)$.

5.10 同上题,设 n 个任务的标号分别为 $1, 2, \cdots, n$,k 个处理器标号分别为 $1, 2, \cdots, k$. 算法基本步骤如下:

1. 如上题,给出将 n 个任务分到 k 个处理器上的分配方法.

2. 在每个分配方法下,找到每个处理器上所分配到的任务并计算该分配方法的罚款.

2.1 设某个处理器上分配到的所有任务是 $w_1, w_2, \cdots, w_u$,用回溯算法计算出以不同顺序在该处理器上执行任务 $w_1, w_2, \cdots, w_u$ 时,每个任务是否超时以及超时的罚款数,再得出以该顺序执行这些任务时的总罚款数,最后找到总罚款数最小的执行顺序. 该回溯算法执行过程如下:可行解是 $w_1, w_2, \cdots, w_u$ 的一个排列,因此解空间是一棵排列树,与根结点关联的 u 条边表示从任务 $w_1, w_2, \cdots, w_u$ 中选择哪个任务执行;对于第一层每个结点,向下分解出 $u-1$ 条边,表示从剩下的 $u-1$ 任务中选择哪一个执行,以此类推. 在某个结点处,如果从根结点到该结点的路径中边的标号顺序为 $w_{i_1}, w_{i_2}, \cdots, w_{i_v}$,则任务 w_{i_v} 的执行时间为 $t_{w_{i_1}} + t_{w_{i_2}} + \cdots + t_{w_{i_v}}$,若 $t_{w_{i_1}} + t_{w_{i_2}} + \cdots + t_{w_{i_v}} > B_{w_{i_1}}$,则增加罚款 $f_{w_{i_v}}$. 在每个叶结点处即能得

到在该顺序下执行 $w_1, w_2, \cdots, w_u$ 的罚款数.遍历整棵树后计算出所有叶结点的罚款数,在所有这些罚款数中找到最小值和相应的执行顺序(最优执行顺序),则最小值即为该处理器执行 $w_1, w_2, \cdots, w_u$ 时罚款的最小值.

2.2 将每个处理器罚款的最小值相加,即为该分配方法按最优顺序执行时的罚款值.合并每个处理器按上述顺序执行的任务,即可得到针对这个分配方法的解.

3. 比较步骤2针对每个分配方法的罚款值,找到其中最小罚款值所对应任务分配方法及其在不同处理器上的执行顺序,即可得到该问题的解.

算法在步骤1的执行时间依然为 $O(nk^n)$;步骤2的执行时间为 $O(n! * n * k^n)$;步骤3的执行 $O(k^n)$.故该算法最坏情形下的时间复杂度为 $O(n * n! * k^n)$.

5.11 本题的解是一个 $m \times n$ 的0-1矩阵 $X[i,j]$,$X[i,j]=1$ 当且仅当陈列室 (i,j) 有哨兵.初始令所有的 $X[i,j]=1, i=1,2,\cdots,m, j=1,2,\cdots,n$.算法从(1,1)开始直到 (m,n) 为止,搜索树是二叉树,有 $m \times n$ 层.每个结点对应一个陈列室位置.如果令 $X[i,j]=0$,取消 (i,j) 位置的哨兵,进入左子树;否则进入右子树.在进入左子树时需要检查房间被监视的情况.即当取消 (i,j) 位置的哨兵时,此位置及其上、下、左、右位置是否被监视.如果有不被监测的情况出现,该分支不再搜索.最坏情况下的时间复杂度为 $O(2^{nm})$.

第 6 章 线性规划

6.1 内容提要

1. 数学模型与基本概念

线性规划的一般形式

$$\min(\max) z = \sum_{j=1}^{n} c_j x_j \qquad \text{目标函数}$$

$$\text{s.t.} \sum_{j=1}^{n} a_{ij} x_j \leqslant (=, \geqslant) b_i, \quad 1 \leqslant i \leqslant m \qquad \text{约束条件}$$

$$x_j \geqslant 0, \quad j \in J \subseteq \{1,2,\cdots,n\} \qquad \text{非负条件}$$

$$x_j \text{ 任意}, \quad j \in \{1,2,\cdots,n\} - J \qquad \text{自由变量}$$

满足约束条件和非负条件的 x 称作**可行解**，使 z 取到最小值(对于 min)或最大值(对于 max)的可行解称作**最优解**.

线性规划的标准形

$$\min z = \sum_{j=1}^{n} c_j x_j$$

$$\text{s.t.} \sum_{j=1}^{n} a_{ij} x_j = b_i \geqslant 0, \quad 1 \leqslant i \leqslant m$$

$$x_j \geqslant 0, \quad 1 \leqslant j \leqslant n$$

标准形的矩阵形式

$$\min z = c^{\mathrm{T}} x$$
$$\text{s.t. } Ax = b$$
$$x \geqslant 0$$

设 A 的秩等于 m，A 的 m 个线性无关的列称作**基**，对应的 m 个变量称作**基变量**，其余的变量称作**非基变量**. 基变量记作 x_B，非基变量记作 x_N. 满足约束条件 $Ax=0$ 且所有非基变量都等于 0 的 x 称作**基本解**，满足非负条件的基本解称作**基本可行解**，对应的基称作**可行基**.

2. 标准形的可行解的性质

主要公式 设可行基 B，则对应的基本可行解 $x_B = B^{-1}b$，$x_N = 0$，简化的目标函数

$$z = z_0 + \lambda^{\mathrm{T}} x$$

其中，$z_0 = c_B^{\mathrm{T}} B^{-1} b$ 是该基本可行解的目标函数值，$\lambda^{\mathrm{T}} = c^{\mathrm{T}} - c_B^{\mathrm{T}} B^{-1} A$ 是**检验数**.

记 $B^{-1}A = (\alpha_{ij})_{m \times n}$，$\beta = B^{-1}b$.

定理 6.1 如果标准形有解，则必有基本可行解.

定理 6.2 如果标准形有最优解，则必存在一个基本可行解是最优解.

定理 6.3 设基本可行解 x，对于 min(max)，如果所有检验数 $\lambda_j \geqslant 0(\lambda_j \leqslant 0)$，$1 \leqslant j \leqslant n$，则 x 是最优解. 如果存在 $\lambda_k < 0(\lambda_k > 0)$ 且所有 $\alpha_{ik} \leqslant 0(1 \leqslant i \leqslant m)$，则目标函数无界.

3. 对偶性

对偶线性规划

原始规划(P)	对偶规划(D)
$\max\ c^{\mathrm{T}} x$	$\min\ b^{\mathrm{T}} y$
s.t. $Ax \leqslant b$	s.t. $A^{\mathrm{T}} y \geqslant c$
$x \geqslant 0$	$y \geqslant 0$

定理 6.4 对偶的对偶是原始规划.

定理 6.5 设 x 和 y 分别是原始规划(P)和对偶规划(D)的可行解，则恒有

$$c^{\mathrm{T}} x \leqslant b^{\mathrm{T}} y$$

定理 6.6 设 x 和 y 分别是原始规划(P)和对偶规划(D)的可行解，如果 $c^{\mathrm{T}} x = b^{\mathrm{T}} y$，则 x 和 y 分别是它们的最优解.

定理 6.7 如果原始规划(P)有最优解，则对偶规划(D)也有最优解，且它们的最优值相等. 反之亦然.

原始规划和对偶规划的解只有下述 3 种可能：

(1) 原始规划和对偶规划都有最优解，且最优值相等.

(2) 原始规划有可行解且目标函数无界，对偶规划无可行解；对偶规划有可行解且目标函数无界，原始规划无可行解.

(3) 原始规划和对偶规划都没有可行解.

定理 6.8(互补松弛性) 设 x 和 y 分别是原始规划(P)和对偶规划(D)的可行解，则 x 和 y 分别是它们的最优解当且仅当

$$\Big(b_i - \sum_{j=1}^{n} a_{ij} x_j\Big) y_i = 0, \quad 1 \leqslant i \leqslant m$$

$$x_j \Big(\sum_{i=1}^{m} a_{ij} y_i - c_j\Big) = 0, \quad 1 \leqslant j \leqslant n$$

4. 算法

单纯形法 取初始可行解 x. 对于 min(max)，若所有 $\lambda_j \geqslant 0(\lambda_j \leqslant 0)$，$1 \leqslant j \leqslant n$，则 x 是最优解，计算结束. 否则取 $\lambda_k < 0(\lambda_k > 0)$，若所有 $\alpha_{ik} \leqslant 0(1 \leqslant i \leqslant m)$，则无最优解，计算结束. 如果存在 $\alpha_{lk} > 0$，则做基变换，重复进行.

两阶段法 阶段一引入人工变量构造辅助问题，用单纯形法解辅助问题. 若辅助问题的最优值 $w^* > 0$，则原问题无可行解，计算结束. 若 $w^* = 0$，则删去人工变量得到原问题的一个可行解，进入阶段二用单纯形法解原问题.

Bland 规则

规则 1：对于 min(max)，当有多个 $\lambda_j<0(\lambda_j>0)$ 时，取其中下标最小的非基变量作为换入变量.

规则 2：当有多个 $\theta_i=\beta_i/\alpha_{ik}(\alpha_{ik}>0)$ 同时取到最小值时，取其中下标最小的基变量作为换出变量.

采用 Bland 规则可避免出现循环，保证单纯形法在有限步内终止.

对偶单纯形法 取一个基本正则解 x（对于 min，$\lambda\geqslant 0$；对于 max，$\lambda\leqslant 0$）. 如果所有 $\beta_i\geqslant 0(1\leqslant i\leqslant m)$，则 x 是最优解，计算结束. 否则取 $\beta_l<0$. 如果所有 $\alpha_{lj}\geqslant 0(1\leqslant j\leqslant n)$，则无可行解，计算结束. 否则做基变换，重复进行.

整数线性规划的分支限界法.

6.2 习 题

6.1 某厂生产两种产品，每件产品的利润分别是 85 元和 70 元. 产品要经过 4 道工序加工，每件产品的加工时间和每周可用的工时如表 6.1 所示.

表 6.1 产品加工时间与用工工时

工序	加工时间/(人·小时/件)		可用工时/(人·小时)
	产品 1	产品 2	
1	0.54	0.85	800
2	0.30	0.70	500
3	1.05	0.55	900
4	0.15	0.25	120

写出下列问题的数学模型：

(1) 制定一周的生产计划，使总利润最大.

(2) 部分工人经过培训掌握 2 个或 3 个工序的操作，从而当需要时可以调剂到其他工序工作. 假设可能调剂的情况如表 6.2 所示.

表 6.2 工序调剂

原工序	调 入 工 序				最大调剂工时/(人·小时)
	1	2	3	4	
1	—	√	√	×	100
2	√	—	×	√	50
3	×	√	—	√	100
4	×	×	√	—	40

试制定一周的生产计划及工序之间的调剂工时，使总利润最大.

6.2 咖啡制造厂用 3 种咖啡豆制造一种混合咖啡，每种咖啡的香味等级、味道等级、售价及库存量如表 6.3 所示.

表 6.3　咖啡豆及库存

咖啡豆	香味等级	味道等级	售价/(元/千克)	库存/千克
1	75	86	20	500
2	85	88	28	600
3	60	75	18	400

假设混合咖啡的香味等级和味道等级是所用咖啡豆的香味等级和味道等级的加权平均值,等级越高质量越好. 现要生产1000千克混合咖啡,要求香味等级不低于75,味道等级不低于80. 要使成本最低应如何配制? 试建立该问题的数学模型.

6.3　某人选择4种基金进行组合投资,咨询师为他提供了如表6.4所示的5种可能的年收益率(%).

表 6.4　基金收益

基金	可　能　性				
	1	2	3	4	5
1	5.06	8.12	8.47	40.23	−18.75
2	12.45	3.22	4.51	−1.53	7.63
3	32.18	14.16	33.64	40.25	−18.09
4	32.02	20.53	12.92	7.14	−5.55

此人采用保守的策略,要求可能的最低收益率最大,应如何确定这4种基金的投资比例? 试建立该问题的数学模型.

6.4　用图解法解下列线性规划.

(1) $\max\ x_1+x_2$

s. t. $x_1\leqslant 5$

$x_2\leqslant 3$

$x_1+3x_2\leqslant 11$

$x_1,x_2\geqslant 0$

(2) $\min\ x_1-x_2$

s. t. $2x_1+3x_2\leqslant 14$

$-x_1+x_2\leqslant 3$

$x_1\leqslant 4$

$x_1,x_2\geqslant 0$

(3) $\min\ 2x_1+x_2$

s. t. $x_1+x_2\geqslant 1$

$x_2\leqslant 2$

$x_1,x_2\geqslant 0$

(4) $\min\ 2x_1-x_2$

s. t. $2x_1+x_2\geqslant 2$

$x_1-x_2\leqslant -3$

$x_1, x_2 \geqslant 0$

(5) $\max\ 3x_1 - 2x_2$

s. t. $x_1 - x_2 \geqslant -1$

$3x_1 + x_2 \leqslant 9$

$x_1 + 2x_2 \geqslant 9$

$x_1, x_2 \geqslant 0$

6.5 写出下述线性规划的标准形.

$$\begin{aligned}
\max\ & 3x_1 - 2x_2 + x_3 \\
\text{s. t.}\ & x_1 + 2x_2 - x_3 \leqslant 1 \\
& 4x_1 - 2x_3 \geqslant 5 \\
& x_2 - 5x_3 \leqslant -4 \\
& x_1 - 3x_2 + 2x_3 = -10 \\
& x_1 \geqslant 0,\ x_2 \text{ 任意},\ x_3 \geqslant 0
\end{aligned}$$

6.6 设线性规划

$$\begin{aligned}
\max\ & 2x_1 + x_2 \\
\text{s. t.}\ & -x_1 + 2x_2 \leqslant 4 \\
& x_1 \leqslant 5 \\
& x_1, x_2 \geqslant 0
\end{aligned}$$

(1) 画出它的可行域,用图解法求最优解.

(2) 写出它的标准形,列出所有的基,指出哪些是可行基. 通过列出所有的可行解及其目标函数值找到最优解. 指出每个可行解对应的可行域的顶点.

6.7 设线性规划标准形

$$\begin{aligned}
\max\ & 5x_1 - 2x_2 \\
\text{s. t.}\ & 3x_1 + 2x_2 + x_3 = 10 \\
& x_1 + x_4 = 3 \\
& x_2 + x_5 = 2 \\
& x_j \geqslant 0,\ 1 \leqslant j \leqslant 5
\end{aligned}$$

下列 4 个 x 都满足等式约束,问其中哪些是可行解? 哪些是基本解? 哪些是基本可行解?

$$x^{(1)} = (3, 2, -3, 0, 0)$$

$$x^{(2)} = (3, 0, 1, 0, 2)$$

$$x^{(3)} = (2, 1, 2, 1, 1)$$

$$x^{(4)} = (0, 0, 10, 3, 2)$$

6.8 用图解法和单纯形法解下述线性规划,并指出每张单纯形表中的基本可行解所对应的可行域的顶点.

$$\begin{aligned}
\max\ & x_1 + 2x_2 \\
\text{s. t.}\ & x_1 - x_2 \geqslant -4
\end{aligned}$$

$$x_2 \leqslant 6$$
$$x_1 + x_2 \leqslant 10$$
$$x_1 - x_2 \leqslant 4$$
$$x_1, x_2 \geqslant 0$$

6.9 用单纯形法解习题 6.4 中的线性规划.

6.10 用单纯形法解下列线性规划

(1) min $-2x_1+x_2-x_3$

s.t. $2x_1+x_2 \leqslant 10$

$-4x_1-2x_2+3x_3 \leqslant 10$

$x_1-2x_2+x_3 \leqslant 14$

$x_j \geqslant 0,\ j=1,2,3$

(2) min $x_1+x_2-3x_3$

s.t. $x_1+x_2-2x_3 \leqslant 9$

$x_1+x_2-x_3 \leqslant 2$

$-x_1+x_2+x_3 \leqslant 4$

$x_j \geqslant 0,\ j=1,2,3$

(3) max $3x_1+5x_2+4x_3$

s.t. $2x_1+3x_2+x_3 \leqslant 9$

$-x_1+2x_2+x_3=12$

$3x_1+x_2+x_3 \geqslant 5$

$x_j \geqslant 0,\ j=1,2,3$

(4) min $3x_1-x_2-2x_3+x_4$

s.t. $2x_1+3x_2-x_3-x_4=6$

$-x_1-2x_2+3x_3-4x_4=-8$

$3x_1+4x_2+x_3-6x_4=4$

$x_j \geqslant 0,\ 1 \leqslant j \leqslant 4$

(5) min $x_1+2x_2+3x_3$

s.t. $x_1-2x_2+4x_3=4$

$4x_1-9x_2+14x_3=16$

$x_j \geqslant 0,\ j=1,2,3$

6.11 设线性规划

$$\min\ c^{\mathrm{T}}x$$
$$\text{s.t. } Ax = b \geqslant 0$$
$$x \geqslant 0$$

的基本可行解 x^* 的所有非基变量的检验数都大于 0,证明 x^* 是唯一的最优解.

6.12 表 6.5 最终单纯形表(最小化)给出一个最优的基本可行解 $x=(3,2,0,4,0)$,试通过基变换找到另一个最优的基本可行解,进而给出无穷多个最优解.

表 6.5 习题 6.12 最终单纯形表(最小化)

			-1	-1	0	0	0	θ
c_B	x_B	b	x_1	x_2	x_3	x_4	x_5	
-1	x_1	3	1	0	1	0	-1	
0	x_4	4	0	0	3	1	-8	
-1	x_2	2	0	1	-1	0	2	
	$-z$	5	0	0	0	0	1	

6.13 表 6.6 是一张最终单纯形表(最小化),能否判断它是否有无穷多个最优解?若能,请给出你的结论.

表 6.6 习题 6.13 最终单纯形表(最小化)

			1	-1	0	0	θ
c_B	x_B	b	x_1	x_2	x_3	x_4	
-1	x_2	2	-1	1	1	0	
0	x_4	10	-3	0	-4	0	
	$-z$	2	0	0	1	0	

6.14 写出下列线性规划的对偶.

$$\begin{aligned}
\max\ & 3x_1-2x_2+x_3+4x_4\\
\text{s.t.}\ & x_1+x_2-x_3-x_4\leqslant 6\\
& x_1-2x_2+x_3\geqslant 5\\
& 2x_1+x_2-3x_3+x_4=-4\\
& x_1,x_2,x_3\geqslant 0\ \text{且}\ x_4\ \text{任意}
\end{aligned}$$

6.15 写出习题 6.4(1)和(2)中线性规划的对偶. 根据互补松弛性,利用原始规划的最优解求对偶规划的最优解,并用目标函数值验证它们确实都是最优解.

6.16 线性拟合. 设 $y=ax+b$,现有一组 x 和 y 的实验数据 (x_i,y_i),$1\leqslant i\leqslant n$,要确定 a,b 使得 $|ax_i+b-y_i|(1\leqslant i\leqslant n)$ 的最大值最小.

(1) 写出这个问题的线性规划模型(P).

(2) 写出(P)的对偶(D).

(3) 比较用单纯形法解(P)和(D),说明为什么应该采用(D).

6.17 用对偶单纯形法解下列线性规划.

(1) $$\begin{aligned}
\min\ & x_1+2x_2+3x_3\\
\text{s.t.}\ & 2x_1-x_2+3x_3\geqslant 6\\
& x_1+2x_2+x_3\geqslant 4\\
& x_1,x_2,x_3\geqslant 0
\end{aligned}$$

(2) $$\begin{aligned}
\min\ & 3x_1+x_2+x_3\\
\text{s.t.}\ & x_1+x_2+x_3\leqslant 8\\
& x_1-x_2\geqslant 4\\
& x_2-x_3\geqslant 3
\end{aligned}$$

$x_1, x_2, x_3 \geqslant 0$

6.18 在算法 6.2 对偶单纯形法中,为什么只有得到最优解和无可行解两者情况,而没有目标函数值无界(无最优解)的情况?

6.19 要用 7m 长的钢管截成 3m、2.1m 和 1.5m 的管材各 100 根,如何截取才能使得所用的钢管数最少?试建立该问题的数学模型.

6.20 某厂用 3 种原料生产 3 种产品,原料的消耗、利润、每周的最高产量和原料的最大供应量如表 6.7 所示.

表 6.7 习题 6.20 产品与原料的有关数据

产品	原料消耗/(吨/吨)			利润/(万元/吨)	最高产量/吨
	Ⅰ	Ⅱ	Ⅲ		
1	0.6	0.3	0.1	6	60
2	0.4	0.3	0.3	4	30
3	0.3	0	0.7	5	80
供应量/吨	30	12	25		

除此之外,还有固定成本. 固定成本只与是否生产该产品有关,而与生产量无关. 只要生产该产品,启动相关设备,就有固定成本. 3 种产品的固定成本分别为 30 万元、50 万元和 20 万元. 如何安排生产才能使利润最大?试建立该问题的数学模型.

6.21 用分支限界法解下述整数线性规划 ILP(用图解法解有关的线性规划).

$$\begin{aligned}\max\ & 3x + y \\ \text{s.t.}\ & 5x - 2y \leqslant 17.5 \\ & -x + 2y \leqslant 6 \\ & 3x + 5y \leqslant 26 \\ & x, y \geqslant 0,\ \text{整数}\end{aligned}$$

6.22 用单纯形法或对偶单纯形法解习题 6.21 中 ILP 的松弛和它的两个子问题.

6.3 习题解答与分析

6.1 (1) 设产品 1 和产品 2 一周的产量分别为 x_1 和 x_2,问题可表述为

$$\begin{aligned}\max\ & 85x_1 + 70x_2 \\ \text{s.t.}\ & 0.54x_1 + 0.85x_2 \leqslant 800 \\ & 0.30x_1 + 0.70x_2 \leqslant 500 \\ & 1.05x_1 + 0.55x_2 \leqslant 900 \\ & 0.15x_1 + 0.25x_2 \leqslant 120 \\ & x_1,\ x_2 \geqslant 0\end{aligned}$$

(2) 又设调剂后 4 道工序的可用工时分别为 u_1, u_2, u_3 和 u_4,工序 i 调剂到工序 j 的工时为 u_{ij},$1 \leqslant i, j \leqslant 4$.

$$\max\ 85x_1 + 70x_2$$

$$
\begin{aligned}
\text{s.t.}\ & 0.54x_1 + 0.85x_2 \leqslant u_1 \\
& 0.30x_1 + 0.70x_2 \leqslant u_2 \\
& 1.05x_1 + 0.55x_2 \leqslant u_3 \\
& 0.15x_1 + 0.25x_2 \leqslant u_4 \\
& u_1 = 800 - u_{12} - u_{13} + u_{21} \\
& u_2 = 500 - u_{21} - u_{24} + u_{12} + u_{32} \\
& u_3 = 900 - u_{32} - u_{34} + u_{13} + u_{43} \\
& u_4 = 120 - u_{43} + u_{24} + u_{34} \\
& u_{12} + u_{13} \leqslant 100 \\
& u_{21} + u_{24} \leqslant 50 \\
& u_{32} + u_{34} \leqslant 100 \\
& u_{43} \leqslant 40 \\
& x_1,\ x_2 \geqslant 0,\ u_i \geqslant 0,\ 1 \leqslant i \leqslant 4 \\
& u_{12}, u_{13}, \cdots, u_{43} \geqslant 0
\end{aligned}
$$

整理后得到：

$$
\begin{aligned}
\max\ & 85x_1 + 70x_2 \\
\text{s.t.}\ & 0.54x_1 + 0.85x_2 - u_1 \leqslant 0 \\
& 0.30x_1 + 0.70x_2 - u_2 \leqslant 0 \\
& 1.05x_1 + 0.55x_2 - u_3 \leqslant 0 \\
& 0.15x_1 + 0.25x_2 - u_4 \leqslant 0 \\
& u_1 + u_{12} + u_{13} - u_{21} = 800 \\
& u_2 + u_{21} + u_{24} - u_{12} - u_{32} = 500 \\
& u_3 + u_{32} + u_{34} - u_{13} - u_{43} = 900 \\
& u_4 + u_{43} - u_{24} - u_{34} = 120 \\
& u_{12} + u_{13} \leqslant 100 \\
& u_{21} + u_{24} \leqslant 50 \\
& u_{32} + u_{34} \leqslant 100 \\
& u_{43} \leqslant 40 \\
& x_1, x_2 \geqslant 0,\ u_i \geqslant 0,\ 1 \leqslant i \leqslant 4 \\
& u_{12}, u_{13}, \cdots, u_{43} \geqslant 0
\end{aligned}
$$

6.2 设 3 种咖啡豆的比例分别是 x_1,x_2和 x_3,问题可表述为

$$
\begin{aligned}
\min\ & 20x_1 + 28x_2 + 18x_3 \\
\text{s.t.}\ & 75x_1 + 85x_2 + 60x_3 \geqslant 75 \\
& 86x_1 + 88x_2 + 75x_3 \geqslant 80 \\
& x_1 + x_2 + x_3 = 1 \\
& 1000x_1 \leqslant 500
\end{aligned}
$$

$$1000x_2 \leqslant 600$$
$$1000x_3 \leqslant 400$$
$$x_1, x_2, x_3 \geqslant 0$$

6.3 设4种基金的投资比例分别为 x_1, x_2, x_3 和 x_4，可能的最低年收益率为 $d(\%)$.

$$\begin{aligned}
&\max d \\
&\text{s.t. } 5.06x_1 + 12.45x_2 + 32.18x_3 + 32.02x_4 \geqslant d \\
&\quad 8.12x_1 + 3.22x_2 + 14.16x_3 + 20.53x_4 \geqslant d \\
&\quad 8.47x_1 + 4.51x_2 + 33.64x_3 + 12.92x_4 \geqslant d \\
&\quad 40.23x_1 - 1.53x_2 + 40.25x_3 + 7.14x_4 \geqslant d \\
&\quad -18.75x_1 + 7.63x_2 - 18.09x_3 - 5.55x_4 \geqslant d \\
&\quad x_1 + x_2 + x_3 + x_4 = 1 \\
&\quad x_1, x_2, x_3, x_4 \geqslant 0,\ d \text{ 任意}
\end{aligned}$$

稍加整理后得到：

$$\begin{aligned}
&\max d \\
&\text{s.t. } d - 5.06x_1 - 12.45x_2 - 32.18x_3 - 32.02x_4 \leqslant 0 \\
&\quad d - 8.12x_1 - 3.22x_2 - 14.16x_3 - 20.53x_4 \leqslant 0 \\
&\quad d - 8.47x_1 - 4.51x_2 - 33.64x_3 - 12.92x_4 \leqslant 0 \\
&\quad d - 40.23x_1 + 1.53x_2 - 40.25x_3 - 7.14x_4 \leqslant 0 \\
&\quad d + 18.75x_1 - 7.63x_2 + 18.09x_3 + 5.55x_4 \leqslant 0 \\
&\quad x_1 + x_2 + x_3 + x_4 = 1 \\
&\quad x_1, x_2, x_3, x_4 \geqslant 0,\ d \text{ 任意}
\end{aligned}$$

6.4 (1) 见图6.1，最优解为点 A.

$$x_1 = 5,\quad x_2 = 2,\quad z = 7.$$

(2) 见图6.2，最优解是线段 AB 上的所有点，有无穷多个解.

$$x_1 = 1 - t,\quad x_2 = 3t + 4(1 - t) = 4 - t,\quad z = -3,\quad 0 \leqslant t \leqslant 1$$

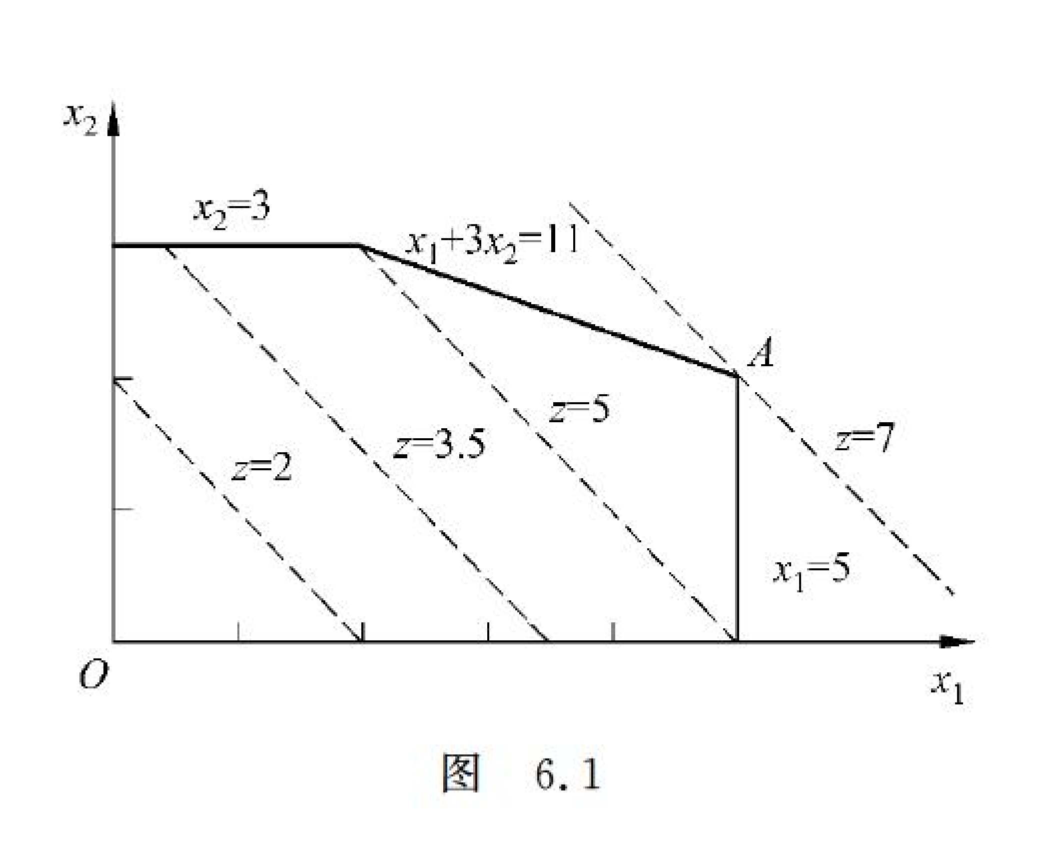


图 6.1

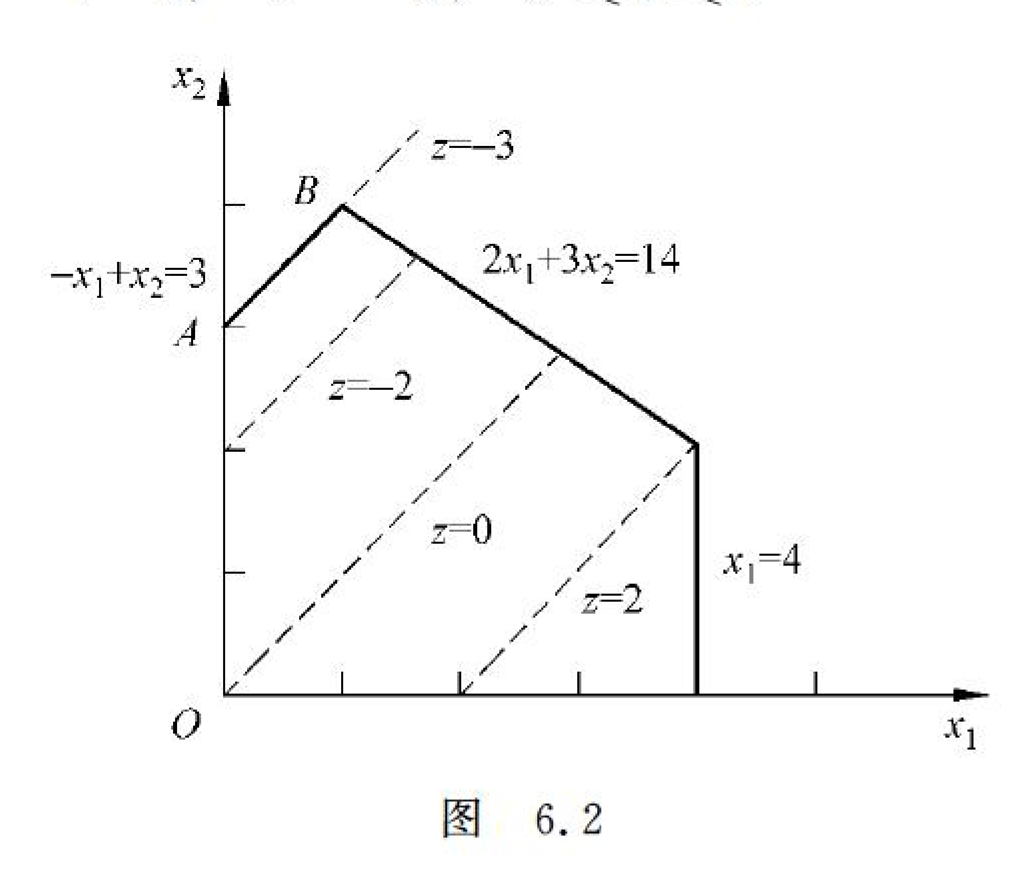


图 6.2

(3) 见图6.3，最优解是点 A.

$$x_1 = 0,\quad x_2 = 1,\quad z = 1.$$

(4) 见图 6.4,有可行解,目标函数无界,无最优解.

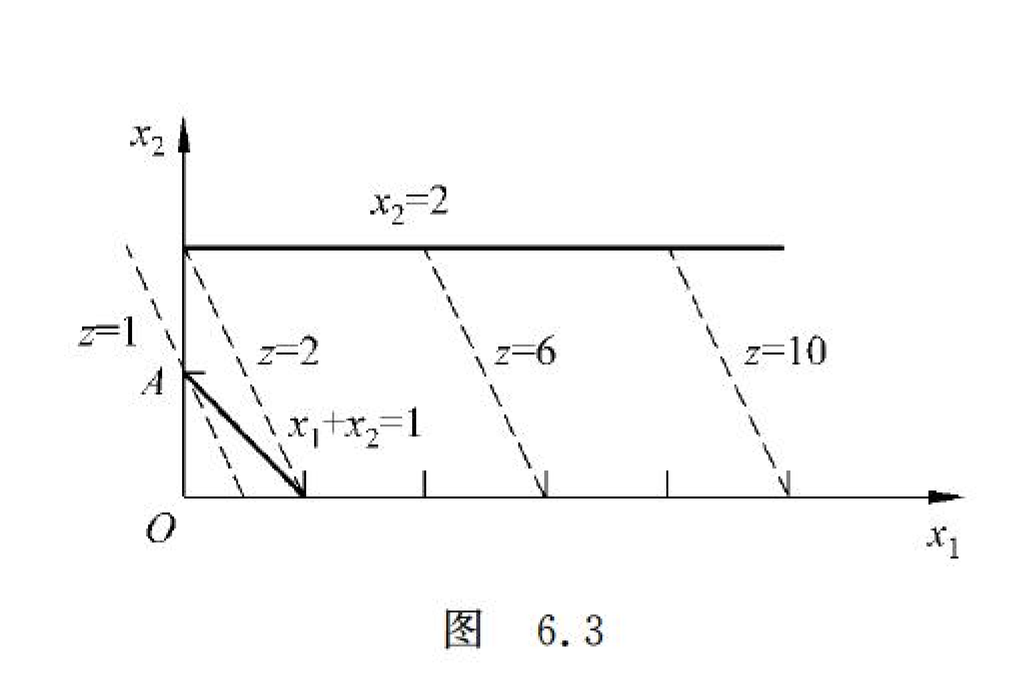


图 6.3

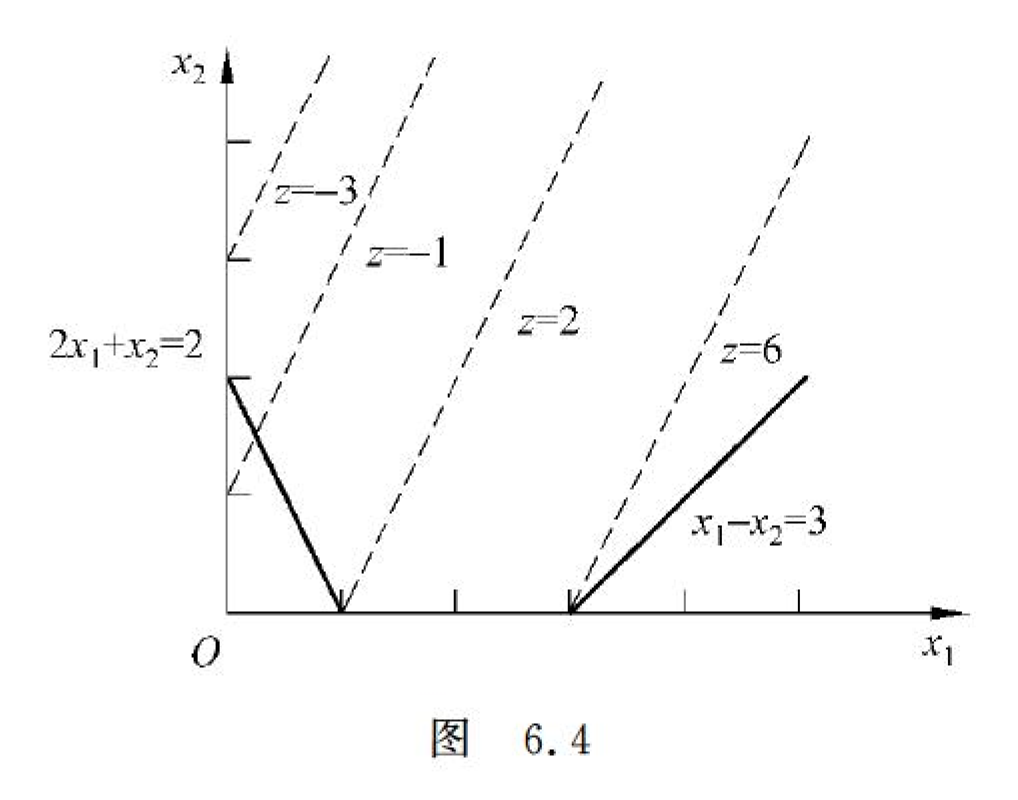


图 6.4

(5) 见图 6.5,无可行解.

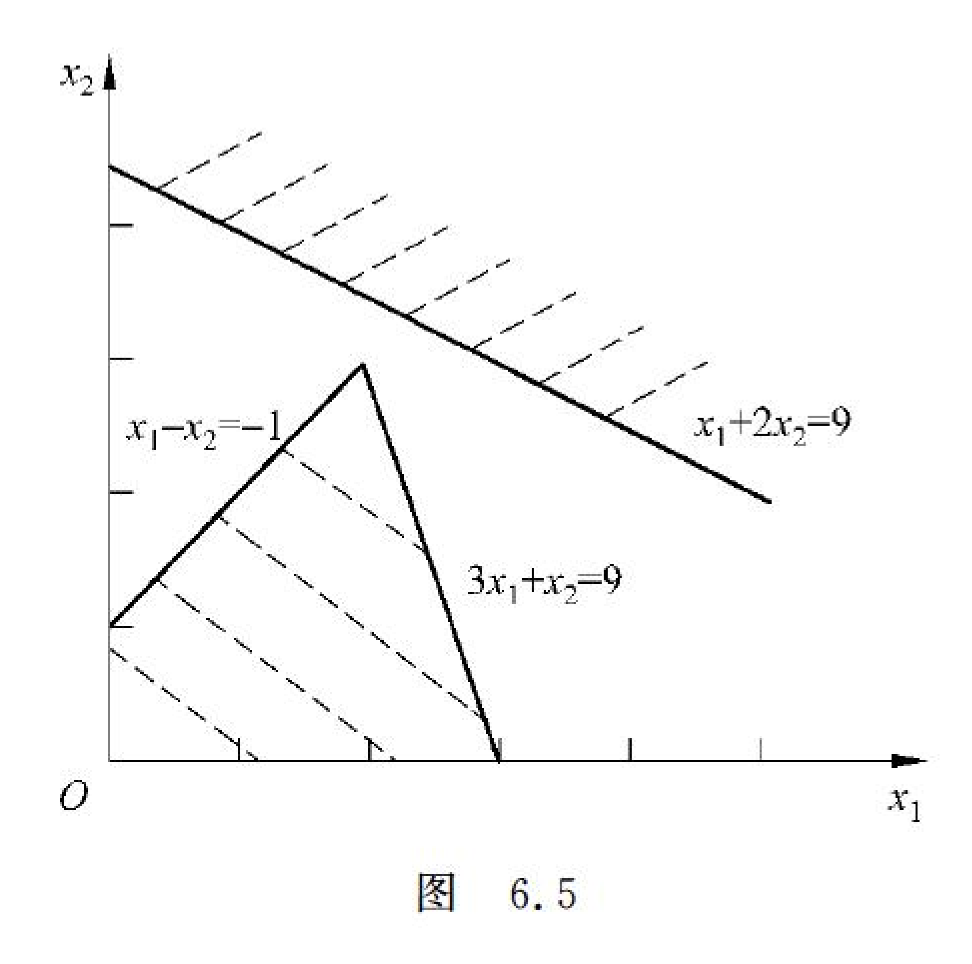


图 6.5

6.5 将 max 改写成 min,第 3 和第 4 式两边变号,并令 $x_2=x_{21}-x_{22}$,

$$
\begin{aligned}
\min\ & -3x_1+2x_{21}-2x_{22}-x_3 \\
\text{s.t.}\ & x_1+2x_{21}-2x_{22}-x_3 \leqslant 1 \\
& 4x_1 \qquad\qquad\qquad -2x_3 \geqslant 5 \\
& \qquad -x_{21}+x_{22}+5x_3 \geqslant 4 \\
& -x_1+3x_{21}-3x_{22}-2x_3 = 10 \\
& x_1,x_{21},x_{22},x_3 \geqslant 0
\end{aligned}
$$

再引入松弛变量 x_4 和剩余变量 x_5,x_6,得到标准形

$$
\begin{aligned}
\min\ & -3x_1+2x_{21}-2x_{22}-x_3 \\
\text{s.t.}\ & x_1+2x_{21}-2x_{22}-x_3+x_4 = 1 \\
& 4x_1 \qquad\qquad\qquad -2x_3-x_5 = 5 \\
& \qquad -x_{21}+x_{22}+5x_3-x_6 = 4 \\
& -x_1+3x_{21}-3x_{22}-2x_3 = 10 \\
& x_1,x_{21},x_{22},x_3,x_4,x_5,x_6 \geqslant 0
\end{aligned}
$$

6.6 (1) 见图6.6,最优解是点B.

$$x_1=5,\quad x_2=4.5,\quad z=14.5.$$

(2) 标准形为

$$\begin{aligned}&\min -2x_1-x_2\\&\text{s.t.}\ -x_1+2x_2+x_3\qquad =4\\&\qquad x_1\qquad\qquad +x_4=5\\&\qquad x_j\geqslant 0,\quad 1\leqslant j\leqslant 4\end{aligned}$$

$$A=\begin{pmatrix}-1&2&1&0\\1&0&0&1\end{pmatrix}$$

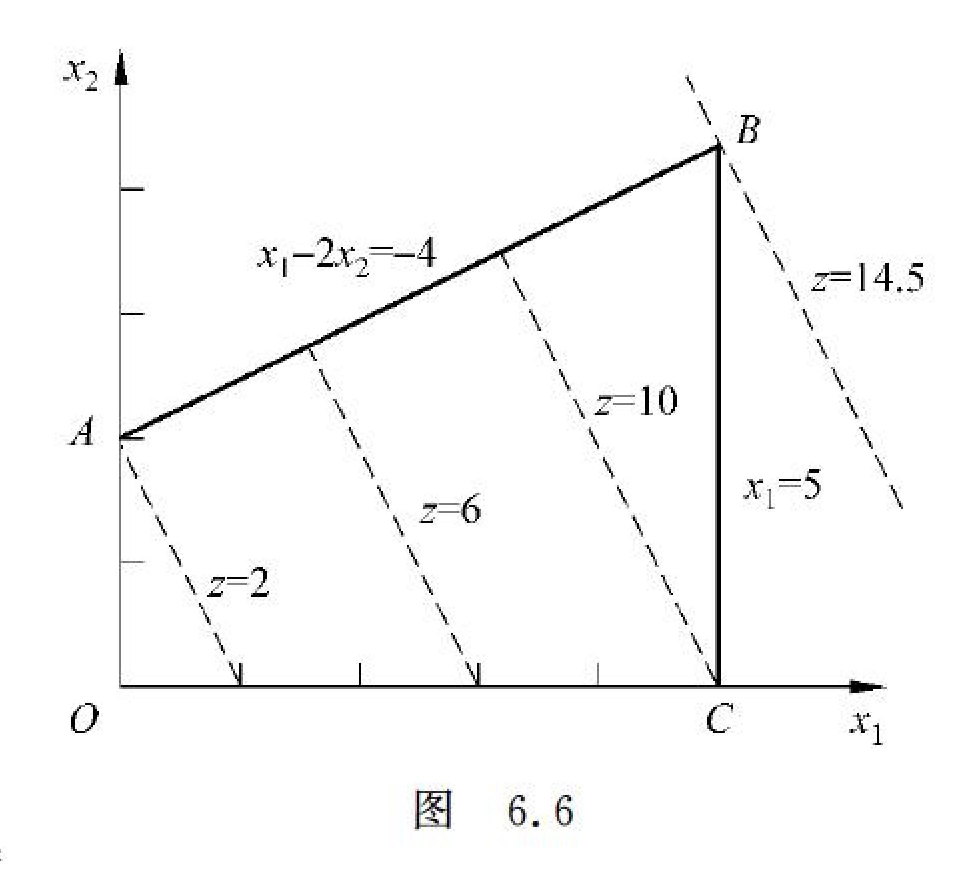


图 6.6

$B_1=(P_1,P_2)=\begin{pmatrix}-1&2\\1&0\end{pmatrix}$,$x_1^{(1)}=5$,$x_2^{(1)}=4.5$,$x_3^{(1)}=0$,$x_4^{(1)}=0$,$z^{(1)}=-14.5$. 对应点$B$. B_1是可行基.

$B_2=(P_1,P_3)=\begin{pmatrix}-1&1\\1&0\end{pmatrix}$,$x_1^{(2)}=5$,$x_2^{(2)}=0$,$x_3^{(2)}=9$,$x_4^{(2)}=0$,$z^{(2)}=-10$. 对应点$C$. B_2是可行基.

$B_3=(P_1,P_4)=\begin{pmatrix}-1&0\\1&1\end{pmatrix}$,$x_1^{(3)}=-4$,$x_2^{(3)}=0$,$x_3^{(3)}=0$,$x_4^{(3)}=9$. B_3是基,但不是可行基.

$B_4=(P_2,P_3)=\begin{pmatrix}2&1\\0&0\end{pmatrix}$,不是基.

$B_5=(P_2,P_4)=\begin{pmatrix}2&0\\0&1\end{pmatrix}$,$x_1^{(5)}=0$,$x_2^{(5)}=2$,$x_3^{(5)}=0$,$x_4^{(5)}=5$,$z^{(5)}=-2$. 对应点$A$,$B_5$是可行基.

$B_6=(P_3,P_4)=\begin{pmatrix}1&0\\0&1\end{pmatrix}$,$x_1^{(5)}=0$,$x_2^{(5)}=0$,$x_3^{(5)}=4$,$x_4^{(5)}=5$,$z^{(5)}=0$. 对应点$O$,$B_6$是可行基.

$x^{(1)}=(5,4.5,0,0)$是最优解.

6.7 $x^{(1)}=(3,2,-3,0,0)$是基本解、但不是可行解,$x^{(3)}=(2,1,2,1,1)$是可行解、但不是基本解,$x^{(2)}=(3,0,1,0,2)$和$x^{(4)}=(0,0,10,3,2)$是基本可行解.

6.8 图解法见图6.7,最优解是点C,

$$x_1=4,\quad x_2=6,\quad z=16.$$

写成标准形

$$\begin{aligned}&\min -x_1-2x_2\\&\text{s.t.}\ -x_1+x_2+x_3\qquad\qquad =4\\&\qquad\quad x_2\quad +x_4\qquad\quad =6\\&\qquad x_1+x_2\qquad +x_5\quad =10\\&\qquad x_1-x_2\qquad\qquad +x_6=4\\&\qquad x_j\geqslant 0,\ 1\leqslant j\leqslant 6\end{aligned}$$

用单纯形法计算见表6.8. 按照表中的顺序,计算中出现的基本可行解如下

$x_1^{(1)}=0$，$x_2^{(1)}=0$，$x_3^{(1)}=4$，$x_4^{(1)}=6$，$x_5^{(1)}=10$，$x_6^{(1)}=4$，$z^{(1)}=0$. 对应点 O.

$x_1^{(2)}=0$，$x_2^{(2)}=4$，$x_3^{(2)}=0$，$x_4^{(2)}=2$，$x_5^{(2)}=6$，$x_6^{(2)}=8$，$z^{(2)}=-8$. 对应点 A.

$x_1^{(3)}=2$，$x_2^{(3)}=6$，$x_3^{(3)}=0$，$x_4^{(3)}=0$，$x_5^{(3)}=2$，$x_6^{(3)}=8$，$z^{(3)}=-14$. 对应点 B.

$x_1^{(4)}=4$，$x_2^{(4)}=6$，$x_3^{(4)}=2$，$x_4^{(4)}=0$，$x_5^{(4)}=0$，$x_6^{(4)}=6$，$z^{(4)}=-16$. 对应点 C. 这是最优解.

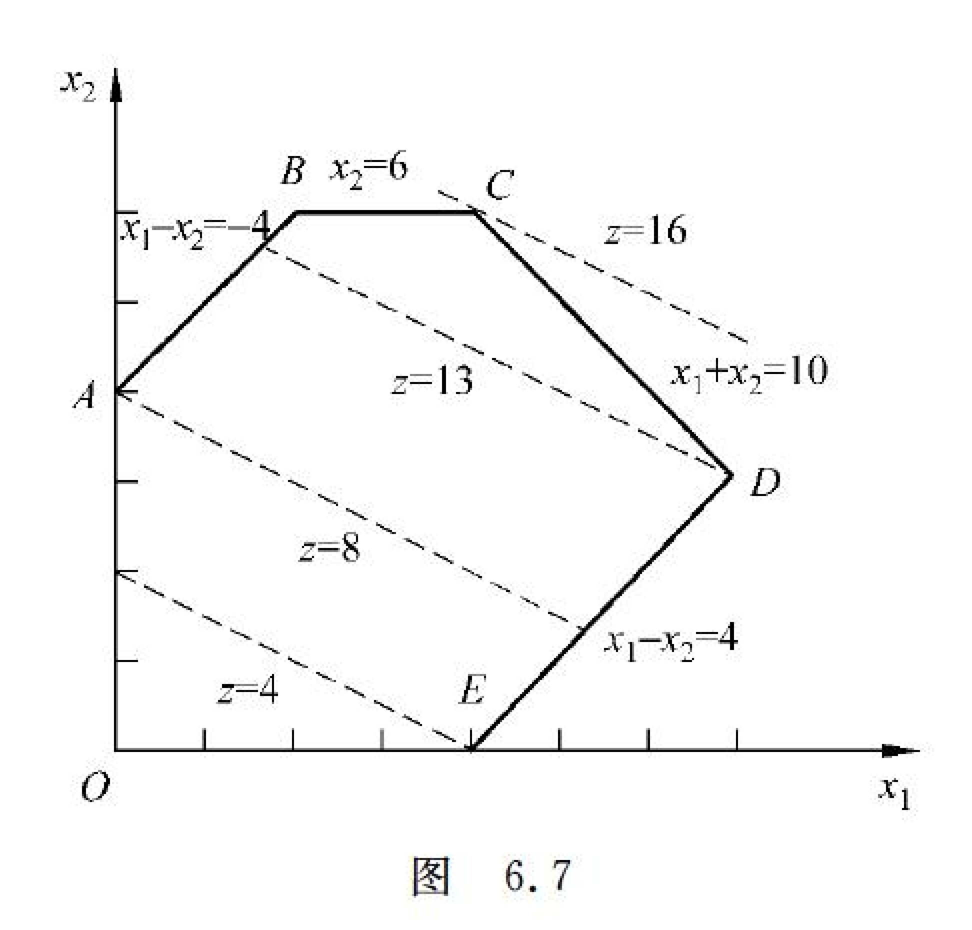


图 6.7

表 6.8

			−1	−2	0	0	0	0	θ
c_B	x_B	b	x_1	x_2	x_3	x_4	x_5	x_6	
0	x_3	4	−1	①	1	0	0	0	4
0	x_4	6	0	1	0	1	0	0	6
0	x_5	11	1	1	0	0	1	0	10
0	x_6	4	1	−1	0	0	0	1	—
	$-z$	0	−1	−2	0	0	0	0	
−2	x_2	4	−1	1	1	0	0	0	—
0	x_4	2	①	0	−1	1	0	0	2
0	x_5	6	2	0	−1	0	1	0	3
0	x_6	8	0	0	1	0	0	1	—
	$-z$	8	−3	0	2	0	0	0	
−2	x_2	6	0	1	0	1	0	0	—
−1	x_1	2	1	0	−1	1	0	0	—
0	x_5	2	0	0	①	−2	1	0	2
0	x_6	8	0	0	1	0	0	1	8
	$-z$	14	0	0	−1	3	0	0	
−2	x_2	6	0	1	0	1	0	0	
−1	x_1	4	1	0	0	−1	1	0	
0	x_3	2	0	0	1	−2	1	0	
0	x_6	6	0	0	0	2	−1	1	
	$-z$	16	0	0	0	1	1	0	

6.9 (1) 化成标准形

$$\begin{aligned} &\min -x_1-x_2\\ &\text{s.t.}\ x_1+x_3=5\\ &\quad x_2+x_4=3\\ &\quad x_1+3x_2+x_5=11\\ &\quad x_j\geqslant 0,\ 1\leqslant j\leqslant 5 \end{aligned}$$

用单纯形法计算见表 6.9. 最优解为

$$x_1=5,\quad x_2=2,\quad z=-7.$$

表 6.9

			-1	-1	0	0	0	θ
c_B	x_B	b	x_1	x_2	x_3	x_4	x_5	
0	x_3	5	①	0	1	0	0	5
0	x_4	3	0	1	0	1	0	—
0	x_5	11	1	3	0	0	1	11
	$-z$	0	-1	-1	0	0	0	
-1	x_1	5	1	0	1	0	0	—
0	x_4	3	0	1	0	1	0	3
0	x_5	6	0	③	-1	0	1	2
	$-z$	5	0	-1	1	0	0	
-1	x_1	5	1	0	1	0	0	
0	x_4	1	0	0	1/3	1	$-1/3$	
-1	x_2	2	0	1	$-1/3$	0	1/3	
	$-z$	7	0	0	2/3	0	1/3	

(2) 化成标准形

$$\begin{aligned} &\min x_1-x_2\\ &\text{s.t.}\ 2x_1+3x_2+x_3=14\\ &\quad -x_1+x_2+x_4=3\\ &\quad x_1+x_5=4\\ &\quad x_j\geqslant 0,\ 1\leqslant j\leqslant 5 \end{aligned}$$

单纯形法的计算过程见表 6.10. 在第 2 张单纯形表中所有检验数都大于等于 0,得到最优解 $x_1^{(1)}=0, x_2^{(1)}=3, z=-3$. 注意到非基变量 x_1 的检验数为 0,可以以 x_1 作为换入变量做基变换,见接下来的单纯形表,得到另一个最优解 $x_1^{(2)}=1, x_2^{(2)}=4$,最优值当然不变 $z=-3$. 从而有无穷多个最优解 $x_1=1-t, x_2=3t+4(1-t)=4-t, 0\leqslant t\leqslant 1$.

$x^{(1)}$ 和 $x^{(2)}$ 分别对应习题 6.4(2)解中的点 A 和点 B.

(3) 引入剩余变量 x_3,松弛变量 x_4 和人工变量 x_5,

$$\begin{aligned} &\min 2x_1+x_2\\ &\text{s.t.}\ x_1+x_2-x_3+x_5=1\\ &\quad x_2+x_4=2 \end{aligned}$$

$$x_j \geqslant 0, 1 \leqslant j \leqslant 5$$

表 6.10

			1	−1	0	0	0	θ
c_B	x_B	b	x_1	x_2	x_3	x_4	x_5	
0	x_3	14	2	3	1	0	0	14/3
0	x_4	3	−1	①	0	1	0	3
0	x_5	4	1	0	0	0	1	—
	$-z$	0	1	−1	0	0	0	
0	x_3	5	⑤	0	1	−3	0	1
−1	x_2	3	−1	1	0	1	0	—
0	x_5	4	1	0	0	0	1	4
	$-z$	3	0	0	0	1	0	
1	x_1	1	1	0	1/5	−3/5	0	
−1	x_2	4	0	1	1/5	2/5	0	
0	x_5	3	0	0	−1/5	3/5	1	
	$-z$	3	0	0	0	1	0	

用两阶段法．阶段一见表 6.11，$w^* = 0$，得到一个基本可行解．删去 x_5 所在列，阶段二见表 6.12，最优解为 $x_1 = 0, x_2 = 1, z = 1$．

表 6.11

			0	0	0	0	1	θ
c_B	x_B	b	x_1	x_2	x_3	x_4	x_5	
1	x_5	1	①	1	−1	0	1	1
0	x_4	2	0	1	0	1	0	—
	$-w$	−1	−1	−1	1	0	0	
1	x_1	1	1	1	−1	0	1	
0	x_4	2	0	1	0	1	0	
	$-w$	0	0	0	0	0	1	

表 6.12

			2	1	0	0	θ
c_B	x_B	b	x_1	x_2	x_3	x_4	
2	x_1	1	1	①	−1	0	1
0	x_4	2	0	1	0	1	2
	$-z$	−2	0	−1	2	0	
1	x_2	1	1	1	−1	0	
0	x_4	1	−1	0	1	1	
	$-z$	−1	1	0	1	0	

(4) 引入剩余变量 x_3 和 x_4,人工变量 x_5 和 x_6,得到

$$
\begin{aligned}
&\min\ 2x_1 - x_2 \\
&\text{s.t.}\quad 2x_1 + x_2 - x_3 \qquad + x_5 \qquad = 2 \\
&\qquad -x_1 + x_2 \qquad - x_4 \qquad + x_6 = 3 \\
&\qquad x_j \geqslant 0,\ 1 \leqslant j \leqslant 6
\end{aligned}
$$

用两阶段法. 阶段一见表 6.13,$w^* = 0$,得到一个基本可行解. 阶段二见表 6.14. 由于 x_4 的检验数小于 0 且所在列的 α_{14} 和 α_{24} 都小于等于 0,故最优值无界,没有最优解.

表 6.13

			0	0	0	0	1	1	θ
c_B	x_B	b	x_1	x_2	x_3	x_4	x_5	x_6	
1	x_5	2	2	①	−1	0	1	0	2
1	x_6	3	−1	1	0	−1	0	1	3
	$-w$	−5	−1	−2	1	1	0	0	
0	x_2	2	2	1	−1	0	1	0	—
1	x_6	1	−3	0	①	−1	−1	1	1
	$-w$	−1	3	0	−1	1	2	0	
0	x_2	3	−1	1	0	−1	0	1	
0	x_3	1	−3	0	1	−1	−1	1	
	$-w$	0	0	0	0	0	1	1	

表 6.14

			2	1	0	0	θ
c_B	x_B	b	x_1	x_2	x_3	x_4	
−1	x_2	3	−1	1	0	−1	
0	x_3	1	−3	0	1	−1	
	$-z$	3	1	0	0	−1	

(5) 引入松弛变量 x_3 和 x_4,剩余变量 x_5,人工变量 x_6,得到

$$
\begin{aligned}
&\min\ -3x_1 + 2x_2 \\
&\text{s.t.}\ -x_1 + x_2 + x_3 \qquad\qquad = 1 \\
&\qquad 3x_1 + x_2 \qquad + x_4 \qquad = 9 \\
&\qquad x_1 + 2x_2 \qquad - x_5 + x_6 = 9 \\
&\qquad x_j \geqslant 0,\ 1 \leqslant j \leqslant 6
\end{aligned}
$$

用两阶段法. 阶段一见表 6.15,由于 $w^* = 1 > 0$,原规划无可行解.

表 6.15

			0	0	0	0	0	1	θ
c_B	x_B	b	x_1	x_2	x_3	x_4	x_5	x_6	
0	x_3	1	−1	①	1	0	0	0	1
0	x_4	9	3	1	0	1	0	0	9
1	x_6	9	1	2	0	0	−1	1	9/2
	$-w$	−9	−1	−2	0	0	0	0	
0	x_2	1	−1	1	1	0	0	0	—
0	x_4	8	④	0	−1	1	0	0	2
0	x_6	7	3	0	−2	0	−1	1	7/3
	$-w$	−7	−3	0	2	0	0	0	
0	x_2	3	0	1	3/4	1/4	0	0	
0	x_1	2	1	0	−1/4	1/4	0	0	
1	x_6	1	0	0	5/4	−3/4	−1	1	
	$-w$	−1	0	0	5/4	3/4	0	0	

6.10 (1) 化成标准形

$$
\begin{aligned}
&\min -2x_1 + x_2 - x_3 \\
&\text{s.t.}\quad 2x_1 + x_2 + x_4 = 10 \\
&\quad -4x_1 - 2x_2 + 3x_3 + x_5 = 10 \\
&\quad x_1 - 2x_2 + x_3 + x_6 = 14 \\
&\quad x_j \geqslant 0,\ 1 \leqslant j \leqslant 6
\end{aligned}
$$

用单纯形法计算见表 6.16，最优解为 $x_1=24/5$，$x_2=2/5$，$x_3=10$，最优值 $z=-96/5$.

表 6.16

			−2	1	−1	0	0	0	θ
c_B	x_B	b	x_1	x_2	x_3	x_4	x_5	x_6	
0	x_4	10	②	1	0	1	0	0	5
0	x_5	10	−4	−2	3	0	1	0	—
0	x_6	14	1	−2	1	0	0	1	14
	$-z$	0	−2	1	−1	0	0	0	
−2	x_1	5	1	1/2	0	1/2	0	0	—
0	x_5	30	0	0	3	2	1	0	10
0	x_6	9	0	−5/2	①	−1/2	0	1	9
	$-z$	10	0	2	−1	1	0	0	
−2	x_1	5	1	1/2	0	1/2	0	0	10
0	x_5	3	0	(15/2)	0	7/2	1	−3	2/5
−1	x_3	9	0	−5/2	1	−1/2	0	1	—

续表

			-2	1	-1	0	0	0	θ
c_B	x_B	b	x_1	x_2	x_3	x_4	x_5	x_6	
	$-z$	19	0	$-1/2$	0	$1/2$	0	1	
-2	x_1	$24/5$	1	0	0	$4/15$	$-1/15$	$1/5$	
1	x_2	$2/5$	0	1	0	$7/15$	$2/15$	$-2/5$	
-1	x_3	10	0	0	1	$2/3$	$1/3$	0	
	$-z$	$96/5$	0	0	0	$11/15$	$1/15$	$4/5$	

(2) 写出它的标准形

$$
\begin{aligned}
\min\ & x_1 + x_2 - 3x_3 \\
\text{s. t.}\ & x_1 + x_2 - 2x_3 + x_4 = 9 \\
& x_1 + x_2 - x_3 + x_5 = 2 \\
& -x_1 + x_2 + x_3 + x_6 = 4 \\
& x_j \geqslant 0,\ 1 \leqslant j \leqslant 6
\end{aligned}
$$

计算见表6.17. x_1的检验数小于0且所在列的$\alpha_{i1}(i=1,2,3)$都小于等于0,目标函数无界,无最优解.

表 6.17

			1	1	-3	0	0	0	θ
c_B	x_B	b	x_1	x_2	x_3	x_4	x_5	x_6	
0	x_4	9	1	1	-2	1	0	0	—
0	x_5	2	1	1	-1	0	1	0	—
0	x_6	4	-1	1	①	0	0	1	4
	$-z$	0	1	1	-3	0	0	0	
0	x_4	17	-1	3	0	1	0	2	
0	x_5	6	0	2	0	0	1	1	
-3	x_1	4	-1	1	1	0	0	1	
	$-z$	12	-2	4	0	0	0	3	

(3) 引入松弛变量x_4,剩余变量x_5,人工变量x_6和x_7,得到

$$
\begin{aligned}
\max\ & 3x_1 + 5x_2 + 4x_3 \\
\text{s. t.}\ & 2x_1 + 3x_2 + x_3 + x_4 = 9 \\
& -x_1 + 2x_2 + x_3 + x_6 = 12 \\
& 3x_1 + x_2 + x_3 - x_5 + x_7 = 5 \\
& x_j \geqslant 0,\ 1 \leqslant j \leqslant 7
\end{aligned}
$$

用两阶段法,阶段一的计算见表6.18,由于最优值$w^*=3>0$,原问题无可行解.

表 6.18

			0	0	0	0	0	1	1	θ
c_B	x_B	b	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
0	x_4	9	2	③	1	1	0	0	0	3
1	x_6	12	−1	2	1	0	0	1	0	6
1	x_7	5	3	1	1	0	−1	0	1	5
	$-w$	−17	−2	−3	−2	0	1	0	0	
0	x_2	3	2/3	1	1/3	1/3	0	0	0	9
1	x_6	6	−7/3	0	1/3	−2/3	0	1	0	18
1	x_7	2	7/3	0	(2/3)	−1/3	−1	0	1	3
	$-w$	−8	0	0	−1	1	1	0	0	
0	x_2	2	−1/2	1	0	1/2	(1/2)	0	−1/2	4
1	x_6	5	−7/2	0	0	−1/2	1/2	1	−1/2	10
0	x_3	3	7/2	0	1	−1/2	−3/2	0	3/2	—
	$-w$	−5	7/2	0	0	1/2	−1/2	0	3/2	
0	x_5	4	−1	2	0	1	1	0	−1	
1	x_6	3	−3	−1	0	−1	0	1	0	
0	x_3	9	2	3	1	1	0	0	0	
	$-w$	−3	3	1	0	1	0	0	1	

(4) 引入 3 个人工变量 x_5,x_6 和 x_7,用两阶段法计算,阶段一见表 6.19. $w^*=0$,但基变量中含人工变量 x_6. x_6所在行非人工变量的 $\alpha_{2j}(1\leqslant j\leqslant 4)$都为 0($\beta_2$必为 0),这表明原来的 3 个等式约束中第 2 个是另外 2 个的线性组合,可以删去. 删去第 2 行和人工变量所在的列进入阶段二,继续计算,见表 6.20. 由于 x_3的检验数等于−2 且所在列的 α 值均小于等于 0,目标函数值无界,无最优解.

表 6.19

			0	0	0	0	1	1	1	θ
c_B	x_B	b	x_1	x_2	x_3	x_4	x_5	x_6	x_7	
1	x_5	6	2	3	−1	−1	1	0	0	2
1	x_6	8	1	2	−3	4	0	1	0	4
1	x_7	4	3	④	1	−6	0	0	1	1
	$-w$	−18	−6	−9	3	3	0	0	0	
1	x_5	3	−1/4	0	−7/4	(7/2)	1	0	−3/4	6/7
1	x_6	6	−1/2	0	−7/2	7	0	1	−1/2	6/7
0	x_2	1	3/4	1	1/4	−3/2	0	0	1/4	—
	$-w$	−9	3/4	0	21/4	−21/2	0	0	9/4	
0	x_4	6/7	−1/14	0	−1/2	1	2/7	0	−3/14	
1	x_6	0	0	0	0	0	−2	1	1	
0	x_2	16/7	9/14	1	−1/2	0	3/7	0	−1/14	
	$-w$	0	0	0	0	0	3	0	0	

表 6.20

			3	−1	−2	1	θ
c_B	x_B	b	x_1	x_2	x_3	x_4	
1	x_4	6/7	−1/14	0	−1/2	1	
−1	x_2	16/7	9/14	1	−1/2	0	
	$-z$	10/7	26/7	0	−2	0	

(5) 引入两个人工变量 x_4 和 x_5,用两阶段法求解. 阶段一的计算见表 6.21. 在第 3 张表中,$\lambda \geqslant 0$ 且 $w^*=0$,但基变量中含人工变量 x_5. 注意到 x_5 所在行中非人工变量的 $\alpha_{2j}(j=1,2,3)$不全为 0,取一个非零的,如 $\alpha_{22}=-1$,以 x_2 为换入变量、x_5 为换出变量做基变换. 变换后,基变量中不再含人工变量. 删去人工变量所在的列,得到原规划的单纯形表,进入阶段二,继续计算,见表 6.22. 最优解为 $x_1=4, x_2=0, x_3=0$,最优值 $z=4$.

表 6.21

			0	0	0	1	1	θ
c_B	x_B	b	x_1	x_2	x_3	x_4	x_5	
1	x_4	4	1	−2	④	1	0	1
1	x_5	16	4	−9	14	0	1	8/7
	$-w$	−20	−5	11	−18	0	0	
0	x_3	1	(1/4)	−1/2	1	1/4	0	4
1	x_5	2	1/2	−2	0	−7/2	1	4
	$-w$	−2	−1/2	2	0	9/2	0	
0	x_1	4	1	−2	4	1	0	
1	x_5	0	0	(−1)	−2	−4	1	
	$-w$	0	0	1	2	5	0	
0	x_1	4	1	0	8	9	−2	
0	x_2	0	0	1	2	4	−1	
	$-w$	0	0	0	0	1	1	

表 6.22

			1	−2	3	θ
c_B	x_B	b	x_1	x_2	x_3	
1	x_1	4	1	0	8	1/2
−2	x_2	0	0	1	②	0
	$-z$	−4	0	0	−1	
1	x_1	4	1	−4	0	
3	x_3	0	0	1/2	1	
	$-z$	−4	0	1/2	0	

6.11 记 $z_0=c^{\mathrm{T}}x^*$，x^* 的非基变量的检验数为 λ_N，则 $z=z_0+\lambda_N x_N$. 由于 $\lambda_N>0$，对任意的 $x_N\neq 0$，$z>z_0$，从而 x^* 是唯一的最优解.

6.12 非基变量 x_3 的检验数等于 0，又 α_{13} 和 α_{23} 大于 0. 取 x_3 作为换入变量，做基变换，见表 6.23. 得到另一个最优的基本可行解 $x'=\left(\frac{5}{3},\frac{10}{3},\frac{4}{3},0,0\right)$，从而有无穷多个最优解

$$\begin{aligned}x^* &= tx+(1-t)x' \\ &= \left[3t+\frac{5}{3}(1-t),2t+\frac{10}{3}(1-t),\frac{4}{3}(1-t),4t,0\right] \\ &= \left(\frac{4}{3}t+\frac{5}{3},-\frac{4}{3}t+\frac{10}{3},-\frac{4}{3}t+\frac{4}{3},4t,0\right),\quad 0\leqslant t\leqslant 1\end{aligned}$$

表 6.23

			−1	−1	0	0	0	θ
c_B	x_B	b	x_1	x_2	x_3	x_4	x_5	
−1	x_1	3	1	0	1	0	−1	3
0	x_4	4	0	0	③	1	−8	4/3
−1	x_2	2	0	1	−1	0	2	
	$-z$	5	0	0	0	0	1	
−1	x_1	5/3	1	0	0	−1/3	5/3	
0	x_3	4/3	0	0	1	1/3	−8/3	
−1	x_2	10/3	0	1	0	1/3	−2/3	
	$-z$	5	0	0	0	0	1	

在习题 6.9(2)中已经看到这种情况.

6.13 由于非基变量 x_1 的检验数 $\lambda_1=0$ 且 α_{11}，α_{21} 都小于等于 0，令 $x_1=\delta$，$x_3=0$，解得 $x_2=2+\delta$，$x_4=10+3\delta$. 当 $\delta\geqslant 0$ 时，(x_1,x_2,x_3,x_4) 是可行解且目标函数值 $z=-2$. 从而有无穷多个最优解

$$x=(\delta,2+\delta,0,10+3\delta)\quad \delta\geqslant 0$$

根据习题 6.11～6.13，关于最优解的个数有下述结论：

① 如果基本可行解 x^* 的所有非基变量的检验数都大于 0(对于最小化)，则 x^* 是唯一的最优解.

② 如果有一个基本可行解所有非基变量的检验数都大于等于 0 且有一个等于 0，则有无穷多个最优解.

6.14 把第 2 个约束不等式改为 $-x_1+2x_2-x_3\leqslant -5$.

原始规划

$$\begin{aligned}&\max\ 3x_1-2x_2+x_3+4x_4\\ &\text{s.t.}\quad x_1+x_2-x_3-x_4\leqslant 6\\ &\qquad -x_1+2x_2-x_3\leqslant -5\\ &\qquad 2x_1+x_2-3x_3+x_4=-4\\ &\qquad x_1,x_2,x_3\geqslant 0,x_4\ \text{任意}\end{aligned}$$

对偶规划

$$\begin{aligned}&\min\ 6y_1-5y_2-4y_3\\ &\text{s.t.}\quad y_1-y_2+2y_3\geqslant 3\\ &\qquad y_1+2y_2+y_3\geqslant -2\\ &\qquad -y_1-y_2-3y_3\geqslant 1\\ &\qquad -y_1+y_3=4\\ &\qquad y_1,y_2\geqslant 0,y_3\ \text{任意}\end{aligned}$$

6.15 (1)

原始规划

$$\max\ x_1+x_2$$
$$\text{s.t.}\ \begin{cases} x_1 \leqslant 5 \\ x_2 \leqslant 3 \\ x_1+3x_2 \leqslant 11 \\ x_1, x_2 \geqslant 0 \end{cases}$$

对偶规划

$$\min\ 5y_1+3y_2+11y_3$$
$$\text{s.t.}\ \begin{cases} y_1+y_3 \geqslant 1 \\ y_2+3y_3 \geqslant 1 \\ y_1, y_2, y_3 \geqslant 0 \end{cases}$$

原始规划的最优解是 $x_1=5, x_2=2$.

互补松弛性如下:

① $(5-x_1)y_1=0$

② $(3-x_2)y_2=0$

③ $(11-x_1-3x_2)y_3=0$

④ $x_1(y_1+y_3-1)=0$

⑤ $x_2(y_2+3y_3-1)=0$

由 $x_2=2$ 和②,推出 $y_2=0$. 由 $x_1=5$ 和④,推出 $y_1+y_3-1=0$. 由 $x_2=2$ 和⑤,推出 $y_2+3y_3-1=0$. 解得 $y_1=\frac{2}{3}, y_3=\frac{1}{3}$.

$x=(5,2)$的目标函数值 $z=5+2=7$, $y=\left(\frac{2}{3},0,\frac{1}{3}\right)$的目标函数值 $w=5\times\frac{2}{3}+3\times0+11\times\frac{1}{3}=7$,两者相等,这验证了 $x=(5,2)$和 $y=\left(\frac{2}{3},0,\frac{1}{3}\right)$分别是原始规划和对偶规划的最优解.

(2) 将原始规划中的 min 改为 max,

原始规划

$$\max\ -x_1+x_2$$
$$\text{s.t.}\ \begin{cases} 2x_1+3x_2 \leqslant 14 \\ -x_1+x_2 \leqslant 3 \\ x_1 \leqslant 4 \\ x_1, x_2 \geqslant 0 \end{cases}$$

对偶规划

$$\min\ 14y_1+3y_2+4y_3$$
$$\text{s.t.}\ \begin{cases} 2y_1-y_2+y_3 \geqslant -1 \\ 3y_1+y_2 \geqslant 1 \\ y_1, y_2, y_3 \geqslant 0 \end{cases}$$

原始规划的最优解是 $x_1=0, x_2=3$.

互补松弛性如下:

① $(14-2x_1-3x_2)y_1=0$

② $(3+x_1-x_2)y_2=0$

③ $(4-x_1)y_3=0$

④ $x_1(2y_1-y_2+y_3+1)=0$

⑤ $x_2(3y_1+y_2-1)=0$

由 $x_1=0, x_2=3$ 和①,推出 $y_1=0$. 由 $x_1=0$ 和③,推出 $y_3=0$. 由 $x_2=3$ 和⑤,推出 $3y_1+y_2-1=0$. 解得 $y_2=1$.

$x=(0,3)$的目标函数值 $z=-0+3=3$, $y=(0,1,0)$的目标函数值 $w=14\times0+3\times1+$

$4\times0=3$,两者相等,这验证了 $x=(0,3)$ 和 $y=(0,1,0)$ 分别是原始规划和对偶规划的最优解.

6.16 (1) 设 $|ax_i+b-y_i|$ 的最大值为 d,有

$$-d\leqslant ax_i+b-y_i\leqslant d,\quad 1\leqslant i\leqslant n$$

于是,问题的线性规划模型为

$$\begin{aligned}&\min\ d\\ &\text{s.t.}\ ax_i+b-y_i\leqslant d,\qquad 1\leqslant i\leqslant n\\ &\qquad ax_i+b-y_i\geqslant -d,\quad 1\leqslant i\leqslant n\\ &\qquad a,b\ \text{任意},\ d\geqslant 0\end{aligned}$$

令 $a=a_1-a_2$,$b=b_1-b_2$,改写为

$$\begin{aligned}&\min\ d\\ &\text{s.t.}\ d-x_ia_1+x_ia_2-b_1+b_2\geqslant -y_i,\quad 1\leqslant i\leqslant n\\ &\qquad d+x_ia_1-x_ia_2+b_1-b_2\geqslant y_i,\quad 1\leqslant i\leqslant n\\ &\qquad a_1,a_2,b_1,b_2,\ d\geqslant 0\end{aligned}\qquad\text{(P)}$$

(2) (P)对偶

$$\begin{aligned}&\max -\sum_{i=1}^{n}y_iu_i+\sum_{i=1}^{n}y_iu_{i+n}\\ &\text{s.t.}\ \sum_{i=1}^{2n}u_i\leqslant 1\\ &\qquad -\sum_{i=1}^{n}x_iu_i+\sum_{i=1}^{n}x_iu_{i+n}\leqslant 0\\ &\qquad \sum_{i=1}^{n}x_iu_i-\sum_{i=1}^{n}x_iu_{i+n}\leqslant 0\\ &\qquad -\sum_{i=1}^{n}u_i+\sum_{i=1}^{n}u_{i+n}\leqslant 0\\ &\qquad \sum_{i=1}^{n}u_i-\sum_{i=1}^{n}u_{i+n}\leqslant 0\\ &\qquad u_i\geqslant 0,1\leqslant i\leqslant 2n\end{aligned}\qquad\text{(D)}$$

(3) 总可以假设 $y_i\geqslant0(1\leqslant i\leqslant n)$. 否则,取 $y_0\leqslant\min\{y_i\}$,令 $y_i'=y_i-y_0\geqslant0(1\leqslant i\leqslant n)$,$y'=y-y_0=ax+b'$,其中 $b'=b-y_0$.

将(P)化成标准形需要将前 n 个约束不等式改为≤并引入 n 个松弛变量,后 n 个约束不等式引入 n 个剩余变量.

$$\begin{aligned}&\min\ d\\ &\text{s.t.}\ -d+x_ia_1-x_ia_2+b_1-b_2+\xi_i=y_i\quad 1\leqslant i\leqslant n\\ &\qquad d+x_ia_1-x_ia_2+b_1-b_2-\xi_{i+n}=y_i\quad 1\leqslant i\leqslant n\\ &\qquad a_1,a_2,b_1,b_2,\quad d\geqslant 0,\quad \xi_i\geqslant 0,\quad 1\leqslant i\leqslant 2n\end{aligned}$$

再引入 n 个人工变量用两阶段法求解,有 $3n+5$ 个变量和 $2n$ 个约束等式.

将(D)化成标准形只需要引入 5 个松弛变量,用单纯形法求解,只有 $2n+5$ 个变量和 5

个约束等式,规模小得多,因此应该使用模型(D). 在最终单纯形表中,5个松弛变量的检验数即为 d, a_1, a_2, b_1, b_2 的最优值.

6.17 (1) 引入变量 x_4, x_5,将问题改写成

$$\begin{aligned}&\min\ x_1+2x_2+3x_3\\&\text{s.t.}\ -2x_1+x_2-3x_3+x_4=-6\\&\quad -x_1-2x_2-x_3+x_5=-4\\&\quad x_j\geqslant 0,\quad 1\leqslant j\leqslant 5\end{aligned}$$

用对偶单纯形法计算见表6.24,初始基是正则基,最优解为 $x_1=16/5, x_2=2/5, x_3=0$,最优值 $z=4$.

表 6.24

			1	2	3	0	0
c_B	x_B	b	x_1	x_2	x_3	x_4	x_5
0	x_4	−6	(−2)	1	−3	1	0
0	x_5	−4	−1	−2	−1	0	1
	$-z$	0	1	2	3	0	0
	θ		−1/2	—	−1	—	—
1	x_1	3	1	−1/2	3/2	−1/2	0
0	x_5	−1	0	(−5/2)	1/2	−1/2	1
	$-z$	−3	0	5/2	3/2	1/2	0
	θ		—	−1	—	−1	—
1	x_1	16/5	1	0	7/5	−2/5	−1/5
2	x_5	2/5	0	1	−1/5	1/5	−2/5
	$-z$	−4	0	0	2	0	1

(2) 引入变量 x_4, x_5 和 x_6,问题可改写成

$$\begin{aligned}&\min\ 3x_1+x_2+x_3\\&\text{s.t.}\ x_1+x_2+x_3+x_4=8\\&\quad -x_1+x_2+x_5=-4\\&\quad -x_2+x_3+x_6=-3\\&\quad x_j\geqslant 0,\ 1\leqslant j\leqslant 6\end{aligned}$$

用对偶单纯形法计算见表6.25. 由于 $\beta_1=-2$,而所有 $\alpha_{1j}\geqslant 0(1\leqslant j\leqslant 6)$,故无可行解.

表 6.25

			3	1	1	0	0	0
c_B	x_B	b	x_1	x_2	x_3	x_4	x_5	x_6
0	x_4	8	1	1	1	1	0	0
0	x_5	−4	(−1)	1	0	0	1	0
0	x_6	−3	0	−1	1	0	0	1

续表

			3	1	1	0	0	0
c_B	x_B	b	x_1	x_2	x_3	x_4	x_5	x_6
	$-z$	0	3	1	1	0	0	0
	θ		-3	—	—	—	—	—
0	x_4	4	0	2	1	1	1	0
3	x_1	4	1	-1	0	0	-1	0
0	x_6	-3	0	(-1)	1	0	0	1
	$-z$	-12	0	4	1	0	3	0
	θ		—	-4	—	—	—	—
0	x_4	-2	0	0	3	1	1	2
3	x_1	7	1	0	-1	0	-1	-1
1	x_2	3	0	1	-1	0	0	-1
	$-z$	-24	0	0	5	0	3	4

6.18 对偶单纯形法的初始基是正则基，对应的初始基本解是对偶规划的可行解. 当对偶规划有可行解时，原始规划只可能是有最优解和无可行解两种情况，不可能出现目标函数值无界的情况.

6.19 每根钢管有下面 7 种截取方式：

① 2 根 3m.

② 1 根 3m，1 根 2.1m，1 根 1.5m.

③ 1 根 3m，2 根 1.5m.

④ 3 根 2.1m.

⑤ 2 根 2.1m，1 根 1.5m.

⑥ 1 根 2.1m，3 根 1.5m.

⑦ 4 根 1.5m.

设采用上述 7 种方式截取的钢管数分别为 x_j，$1\leqslant j\leqslant 7$，问题可表述为下述整数线性规划：

$$
\begin{aligned}
\min\ & x_1+x_2+x_3+x_4+x_5+x_6+x_7\\
\text{s.t.}\ & 2x_1+x_2+x_3 \geqslant 100\\
& x_2+3x_4+2x_5+x_6 \geqslant 100\\
& x_2+2x_3+x_5+3x_6+4x_7 \geqslant 100\\
& x_j\geqslant 0,\ \text{整数},1\leqslant j\leqslant 7
\end{aligned}
$$

6.20 设 3 种产品分别生产 x_1,x_2,x_3（吨），

$$
y_i=\begin{cases}1, & \text{若生产产品 } i,\\ 0, & \text{否则,}\end{cases}\quad i=1,2,3
$$

问题可表成下述混合整数线性规划：

$$
\max\ 6x_1+4x_2+5x_3-30y_1-50y_2-20y_3
$$

$$
\begin{aligned}
\text{s.t. } & 0.6x_1 + 0.4x_2 + 0.3x_3 \leqslant 30 \\
& 0.3x_1 + 0.3x_2 \leqslant 12 \quad \text{原料限制} \\
& 0.1x_1 + 0.3x_2 + 0.7x_3 \leqslant 25 \\
& x_1 \leqslant 60y_1 \\
& x_2 \leqslant 30y_2 \quad \text{产量限制} \\
& x_3 \leqslant 80y_3 \\
& x_1, x_2, x_3 \geqslant 0, \quad y_1, y_2, y_3 = 0, 1
\end{aligned}
$$

正如本题所表明的,0-1 变量在实际应用中常常起到特殊的作用.

6.21 记 ILP 的松弛为 LP,

$$
\begin{aligned}
& \max\ 3x + y \\
\text{s.t. } & 5x - 2y \leqslant 17.5 \\
& -x + 2y \leqslant 6 \\
& 3x + 5y \leqslant 26 \\
& x, y \geqslant 0
\end{aligned}
$$

用图解法得到最优解 $x^{(0)}=4.5, y^{(0)}=2.5$,最优值 $z^{(0)}=16$,在图 6.8 中是点 A.

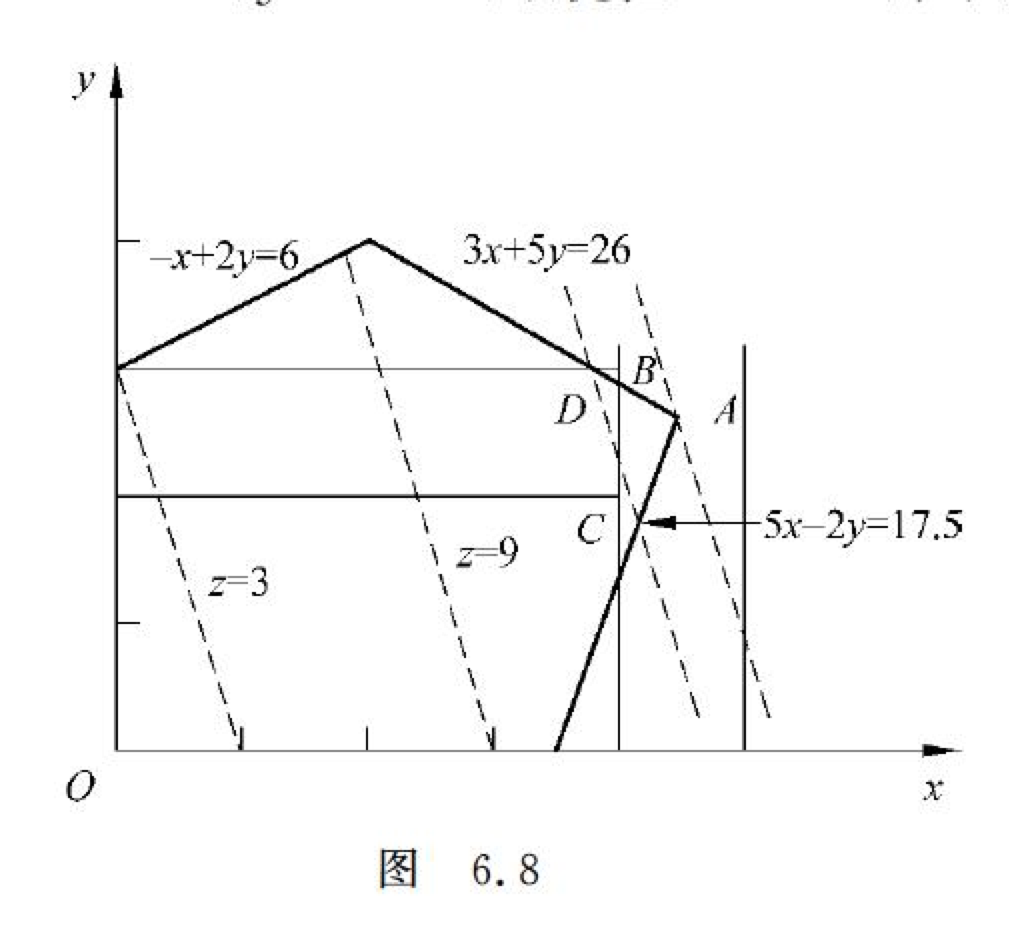


图 6.8

在 LP 上添加 $x\leqslant 4$,记作 LP_1. 由图 6.8,最优解是点 B, $x^{(1)}=4$, $y^{(1)}=2.8$,最优值 $z^{(1)}=14.8$.

在 LP 上添加 $x\geqslant 5$,记作 LP_2,无可行解. 该分支计算结束.

在 LP_1 上添加 $y\leqslant 2$,记作 LP_{11}. 最优解是点 C, $x^{(11)}=4$, $y^{(11)}=2$,最优值 $z^{(11)}=14$. 它是 ILP 的可行解,同时得到 ILP 的最优值的下界 $d=14$. 该分支计算结束.

在 LP_1 上添加 $y\geqslant 3$,记作 LP_{12}. 最优解是点 D, $x^{(12)}=11/3$, $y^{(12)}=3$,最优值 $z^{(12)}=14$. 由于 $z^{(12)}\leqslant d$,该分支计算结束. 至此整个计算结束,ILP 的最优解是点 C, $x^{(11)}=4$, $y^{(11)}=2$,最优值是 $z^{(11)}=14$. 见图 6.9.

6.22 习题 6.21 中 ILP 的松弛 LP 的标准形如下,其中 u_1, u_2, u_3 是新引入的松弛变量,

$$
\begin{aligned}
& \max\ 3x + y \\
\text{s.t. } & 5x - 2y + u_1 = 17.5
\end{aligned}
$$

$$-x + 2y + u_2 = 6$$
$$3x + 5y + u_3 = 26$$
$$x, y, u_1, u_2, u_3 \geqslant 0$$

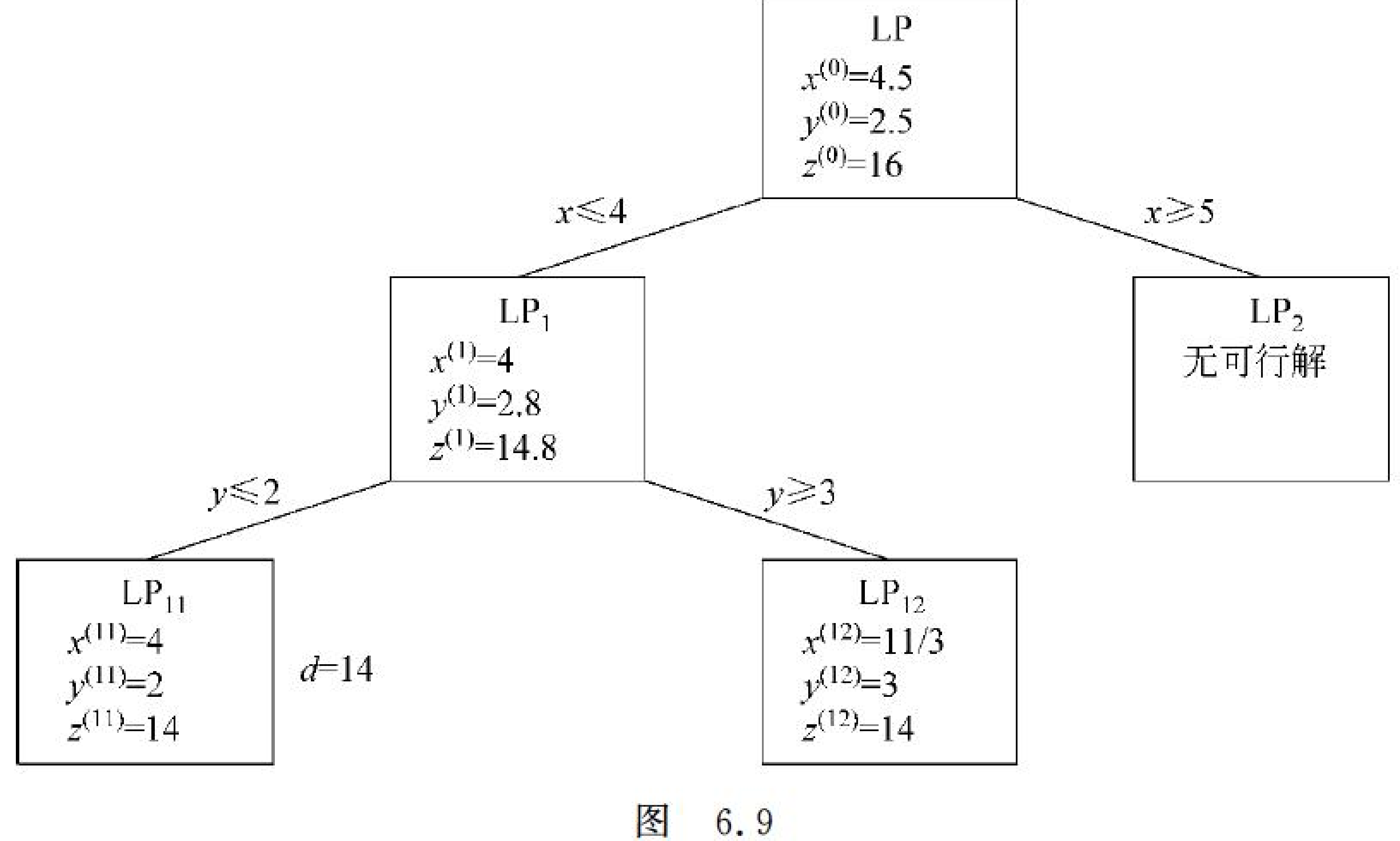


图 6.9

用单纯形法计算见表 6.26，最优解为 $x^{(0)}=4.5$，$y^{(0)}=2.5$. 注意这里是 max，取 $\lambda_k>0$ 的 x_k 作为换入变量，当所有 $\lambda_j\leqslant 0$ 时，得到最优解. 下面两个子问题也是 max，当所有 $\lambda_j\leqslant 0$ 时为正则基.

表 6.26

			3	1	0	0	0	θ
c_B	x_B	b	x	y	u_1	u_2	u_3	
0	u_1	35/2	⑤	−2	1	0	0	7/2
0	u_2	6	−1	2	0	1	0	—
0	u_3	26	3	5	0	0	1	26/3
	$-z$	0	3	1	0	0	0	
3	x	7/2	1	−2/5	1/5	0	0	—
0	u_2	19/2	0	8/5	1/5	1	0	95/16
0	u_3	31/2	0	(31/5)	−3/5	0	1	5/2
	$-z$	−21/2	0	11/5	−3/5	0	0	
3	x	9/2	1	0	5/31	0	2/31	
0	u_2	11/2	0	0	11/31	1	−8/31	
1	y	5/2	0	1	−3/31	0	5/31	
	$-z$	−16	0	0	−12/31	0	−11/31	

子问题 LP_1 由 LP 添加 $x\leqslant 4$ 得到，引入松弛变量 v_1，$x+v_1=4$. 在表 6.26 的最终单纯形表中加入这一行和一列. 新添的行减第 1 行(x 所在行)，消去新添行中的 x 得到基本正则解，见表 6.27 中的第 2 张表. 接下来用对偶单纯形法计算，最优解为 $x^{(1)}=4$，$y^{(1)}=2.8$.

表 6.27

			3	1	0	0	0	0
c_B	x_B	b	x	y	u_1	u_2	u_3	v_1
3	x	9/2	1	0	5/31	0	2/31	0
0	u_2	11/2	0	0	11/31	1	−8/31	0
1	y	5/2	0	1	−3/31	0	5/31	0
		4	1	0	0	0	0	1
	$-z$	−16	0	0	−12/31	0	−11/31	0
3	x	9/2	1	0	5/31	0	2/31	0
0	u_2	11/2	0	0	11/31	1	−8/31	0
1	y	5/2	0	1	−3/31	0	5/31	0
0	v_1	−1/2	0	0	−5/31	0	−2/31	1
	$-z$	−16	0	0	−12/31	0	−11/31	0
	θ		—	—	12/5	—	11/2	—
3	x	4	1	0	0	0	0	1
0	u_2	22/5	0	0	0	1	−2/5	11/5
1	y	14/5	0	1	0	0	1/5	−3/5
0	u_1	31/10	0	0	1	0	2/5	−31/5
	$-z$	−74/5	0	0	0	0	−1/5	−12/5

LP 添加 $x \geqslant 5$ 得到子问题 LP_2,用对偶单纯形法计算见表 6.28. 在初始单纯形表(也是最终单纯形表)中,$\beta_4 < 0$,同时所有 $\alpha_{4j} \geqslant 0 (1 \leqslant j \leqslant 6)$,无可行解.

表 6.28

			3	1	0	0	0	0
c_B	x_B	b	x	y	u_1	u_2	u_3	v_2
3	x	9/2	1	0	5/31	0	2/31	0
0	u_2	11/2	0	0	11/31	1	−8/31	0
1	y	5/2	0	1	−3/31	0	5/31	0
		−5	−1	0	0	0	0	1
	$-z$	−16	0	0	−12/31	0	−11/31	0
3	x	9/2	1	0	5/31	0	2/31	0
0	u_2	11/2	0	0	11/31	1	−8/31	0
1	y	5/2	0	1	−3/31	0	5/31	0
0	v_2	−1/2	0	0	5/31	0	2/31	1
	$-z$	−16	0	0	−12/31	0	−11/31	0

第 7 章 网络流算法

7.1 内容提要

1. 最大流问题

容量网络与可行流

容量网络 $N=<V,E,c,s,t>$，其中$<V,E>$是有向连通图，s，$t\in V$，s 称作**发点**或**源**，t 称作**收点**或**汇**，其余的顶点称作**中间点**. $c:E\to R^*$ 是**容量**.

满足下述条件的 $f:E\to R^*$ 称作 N 上的**可行流**：

(1) 容量限制 $\forall e\in E, f(e)\leqslant c(e)$.

(2) 平衡条件 $\forall i\in V-\{s,t\}$，$\sum_{<j,i>\in E} f(j,i)=\sum_{<i,j>\in E} f(i,j)$.

f 的**流量** $v(f)=\sum_{<s,j>\in E} f(s,j)-\sum_{<j,s>\in E} f(j,s)$. 流量最大的可行流称作**最大流**.

最大流问题 求给定容量网络的最大流.

设 $A\subset V, s\in A, t\in V-A$，称$(A,V-A)=\{<i,j>|<i,j>\in E\wedge i\in A, j\in V-A\}$为**割集**. 割集的**容量** $c(A,V-A)=\sum_{e\in(A,V-A)} c(e)$. 容量最小的割集称作**最小割集**.

增广链

给定容量网络 N 和可行流 f. N 中流量等于容量的边称作**饱和边**，流量小于容量的边称作**非饱和边**，流量为 0 的边称作**零流边**，流量大于 0 的边称作**非零流边**.

不考虑边的方向，从 s 到 t 的边不重复的路径称作 **s-t 链**，规定 s-t 链的方向是从 s 到 t. 链中与链的方向一致的边称作**前向边**，与链的方向相反的边称作**后向边**. 所有前向边都是非饱和的、所有后向边都是非零流的 s-t 链称作 **s-t 增广链**.

如果不存在关于可行流 f 的不含后向边的 s-t 增广链，则称 f 是**极大流**.

辅助网络与分层辅助网络

辅助网络 $N(f)=<V,E(f),ac,s,t>$，其中

$$E^+(f)=\{<i,j>|<i,j>\in E\wedge f(i,j)<c(i,j)\}$$

$$E^-(f)=\{<j,i>|<i,j>\in E\wedge f(i,j)>0\}$$

$$E(f)=E^+(f)\cup E^-(f)$$

$$ac(i,j)=\begin{cases}c(i,j)-f(i,j) & <i,j>\in E^+(f)\\ f(j,i) & <i,j>\in E^-(f)\end{cases}$$

分层辅助网络 $AN(f)=<V(f),AE(f),ac,s,t>$是$N(f)$的子网络,其中

$$V(f)=\sum_{k=0}^{d}V_k(f)$$

$$AE(f)=\bigcup_{k=0}^{d-1}\{<i,j>|<i,j>\in E(f)\text{ 且 } i\in V_k(f),j\in V_{k+1}(f)\}$$

$$V_k(f)=\{i\in V \mid d(i)=k\},\quad 0\leqslant k\leqslant d-1$$

$$V_d(f)=\{t\}$$

这里$d(i)$是在$N(f)$中s到i的距离,$d(s)=0,d(t)=d$.

定理7.1 可行流f是最大流的充分必要条件是不存在关于f的s-t增广链.

定理7.2(最大流最小割集定理) 容量网络中最大流的流量等于最小割集的容量.

Ford-Fulkerson算法(FF算法) 从零流开始.设当前的可行流为f,用标号法找一条关于f的s-t增广链P,修改P上的流量,得到一个流量更大的可行流.重复进行,直到不存在s-t增广链为止.

假设所有容量都是整数,则FF算法在$O(mC)$步内终止,其中m是边数,$C=\sum_{<s,j>\in E}c(s,j)$.

Dinic算法 从零流开始.设当前的可行流为f,构造$AN(f)$,求$AN(f)$上的极大流g.令$f\leftarrow f+g$.重复进行,直到$AN(f)$中s与t不连通为止.

Dinic算法在$O(n^3)$步内终止,其中n是顶点数.

2. 最小费用流

容量-费用网络 $N=<V,E,c,w,s,t>$,其中$w:E\to R^*$是单位费用,其余与容量网络中的相同.

可行流f的**费用**$w(f)=\sum_{e\in E}w(e)f(e)$.所有流量$v_0$的可行流中费用最小的称作流量$v_0$的**最小费用流**.

最小费用流问题 给定容量-费用网络N和流量v_0,求流量v_0的最小费用流.

容量-费用网络N关于可行流f的**辅助网络**$N(f)=<V,E(f),ac,aw,s,t>$,其中辅助费用

$$aw(i,j)=\begin{cases}w(i,j), & <i,j>\in E^+(f)\\ -w(j,i), & <i,j>\in E^-(f)\end{cases}$$

其余与容量网络的辅助网络中的相同.

设C是边不重复的回路,$\delta>0$,C中边上的流量为δ、其余边上的流量为0的可行流称作C上的**圈流**,记作h^C.δ称作h^C的**环流量**.h^C的流量$v(h^C)=0$,费用$w(h^C)=\delta\cdot w(C)$.

定理7.3 流量v_0的可行流f是最小费用流当且仅当$N(f)$中不存在以aw为权的负回路(权小于0的回路).

定理7.4 设f是流量v的最小费用流,P是$N(f)$中权aw的s-t最短路径,g是P上流量θ的可行流,则$f'=f+g$是流量$v+\theta$的最小费用流.

Floyd算法 任给赋权(权可为负数)有向图D,检测D中是否有负回路.如果有负回路,则输出一条负回路.如果没有负回路,则给出任意两点之间的最短路径.

算法采用动态规划方法.记从i到j中间经过顶点号码不大于k的最短路径的长度为

$d^{(k)}(i,j)$,递推公式如下:

$$d^{(0)}(i,j)=w(i,j),\qquad 1\leqslant i,\ j\leqslant n$$
$$d^{(k)}(i,j)=\min\{d^{(k-1)}(i,j),\ d^{(k-1)}(i,k)+d^{(k-1)}(k,j)\}$$
$$1\leqslant i,\ j\leqslant n \text{ 且 } i,\ j\neq k,\ 1\leqslant k\leqslant n$$

当 D 中没有负回路时,$d(i,j)=d^{(n)}(i,j)$. 当 D 中有负回路时,存在 k 和 i,使得 $d^{(k)}(i,i)<0$.

最小费用流的负回路算法 用最大流算法求一个流量 v_0 的可行流 f 作为初始可行流. 设当前的可行流为 f,用 Floyd 算法检测 $N(f)$ 中是否有权 aw 的负回路. 若不存在负回路,则 f 是最小费用流. 若存在负回路 C,则令 $f\leftarrow f+h^C$. 重复进行.

最小费用流的最短路径算法 从零流开始. 设当前可行流为 f,求 $N(f)$ 中以 aw 为权的 s-t 最短路径 P,修改 P 上的流量. 重复进行,直到 $v(f)=v_0$ 为止.

3. 运输问题

运输问题(Hitchcock 问题) 有 m 个产地和 n 个销地,产地 A_i 的产量 a_i,销地 B_j 的销量 b_j,从 A_i 到 B_j 的单位运费 w_{ij},$1\leqslant i\leqslant m$,$1\leqslant j\leqslant n$. 假设产销平衡,$\sum_{i=1}^{m}a_i=\sum_{j=1}^{n}b_j$. 试制订调运方案使得总运费最小.

位势算法 确定初始调运方案. 设当前调运方案 x,计算位势 u_i,v_j 和检验数 $\lambda_{ij}=w_{ij}-u_i-v_j$,$1\leqslant i\leqslant m$,$1\leqslant j\leqslant n$. 若所有 $\lambda_{ij}\geqslant 0$,则 x 是最优调运方案;否则调整调运方案. 重复进行.

表上作业法 以表格的形式进行位势算法的计算.

4. 二部图匹配

匹配

设简单二部图 $G=\langle A,B,E\rangle$,$M\subseteq E$. 如果 M 中任意两条边都不相邻,则称 M 是 G 的**匹配**. 边数最多的匹配称作**最大匹配**. 如果 $|A|=|B|=|M|$,则称 M 是**完美匹配**. M 中的边称作**匹配边**,不属于 M 的边称作**非匹配边**. 与匹配边关联的顶点称作**饱和点**,不与匹配边关联的顶点称作**非饱和点**. G 中起点和终点都是非饱和点、由非匹配边和匹配边交替构成的路径称作**增广交错路径**.

二部图最大匹配问题 求给定二部图的最大匹配.

指派问题 求给定赋权完全二部图的权最小的完美匹配.

定理 7.5 二部图的匹配 M 是最大匹配当且仅当不存在关于 M 的增广交错路径.

二部图最大匹配问题的匈牙利算法 取 $M=\varnothing$ 作为初始匹配. 设当前匹配为 M,用标号法找一条增广交错路径 P,令 $M\leftarrow M\oplus P$. 重复进行,直到不存在增广交错路径为止.

匈牙利算法在 $O(n^3)$ 步内终止.

用 Dinic 算法解二部图最大匹配问题 把二部图最大匹配问题转化成最大流问题,用 Dinic 算法求解. 计算在 $O(n^{1/2}m)$ 步内终止.

指派问题的匈牙利算法 原始-对偶算法的应用,时间复杂度为 $O(n^3)$.

7.2 习 题

7.1 证明:$v(f)=\sum_{\langle j,t\rangle\in E}f(j,t)-\sum_{\langle t,j\rangle\in E}f(t,j)$.

7.2 图7.1给定容量网络 $N=<V,E,c,s,t>$ 和可行流 f_0,其中每条边 e 旁的第1个数是 $c(e)$,第2个数是 $f_0(e)$. 以 f_0 为初始可行流,用FF算法求 N 的最大流 $f_{\max}$ 和最小割集 $(X,V-X)$,并验证 $v(f_{\max})=c(X,V-X)$.

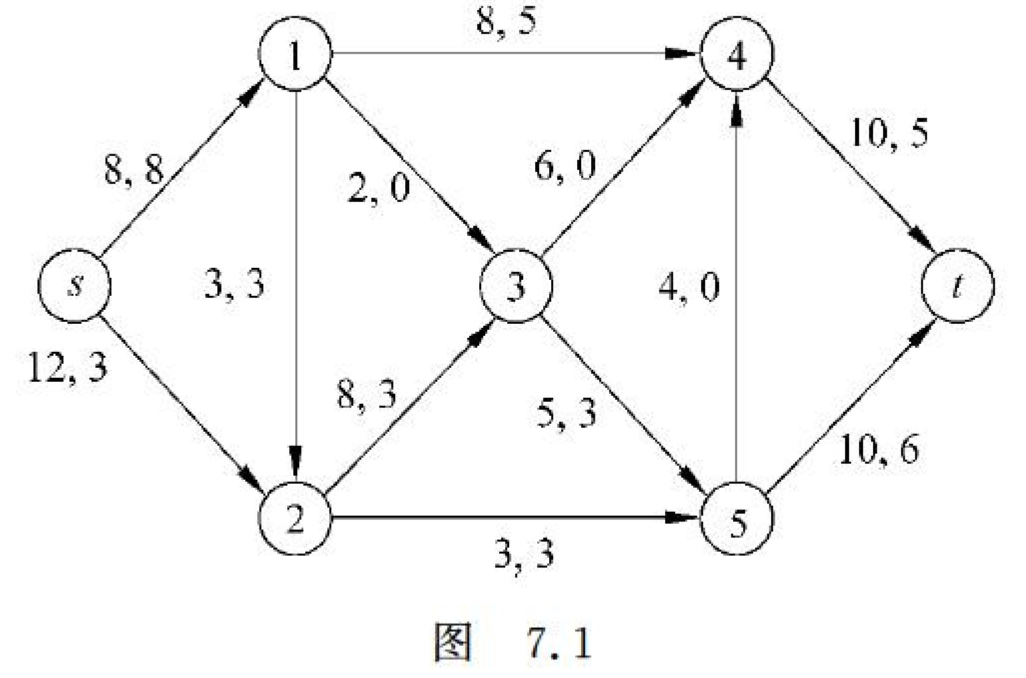


图 7.1

7.3 用Dinic算法求习题7.2中的容量网络 N 的最大流,仍以 f_0 为初始可行流.

7.4 设 N 是简单容量网络,f 是 N 上的0-1可行流,证明 $N(f)$ 也是简单容量网络.

7.5 设计一个算法求容量网络中给定流量的可行流.

7.6 将求多发点、多收点的容量网络的最大流问题转化为标准的最大流问题.

7.7 将求带顶点容量的容量网络的最大流问题转化为标准的最大流问题. 带顶点容量的容量网络 $N=<V,E,c,s,t>$,其中 $c:E\cup(V-\{s,t\})\rightarrow R^*$,即不但每一条边 e 有容量限制 $c(e)$,而且对通过每个中间点的流量也有限制:$\forall u\in V-\{s,t\}$, $\sum\limits_{<v,u>\in E} f(v,u)\leqslant c(u)$.

7.8 设 $N=<V,E,c,s,t>$ 具有单位容量,$|V|=n$,$|E|=m$,N 的最大流量为 v^*. 证明:

(1) N 中 s-t 距离小于等于 $2n/\sqrt{v^*}$.

(2) 对 N 运用Dinic算法的运行时间为 $O(n^{2/3}m)$(提示:参照定理7.4的证明).

7.9 设有向图 $D=<V,E>$,$s,t\in V$,把求从 s 到 t 尽可能多的边不相交的路径问题转化为最大流问题,分别用FF算法和Dinic算法解最大流问题并分析算法的时间复杂度.

7.10 设有向图 $D=<V,E>$,$s,t\in V$,把求从 s 到 t 尽可能多的顶点(除 s,t 外)不相交的路径问题转化为最大流问题,分别用FF算法和Dinic算法解最大流问题并分析算法的时间复杂度.

7.11 用Floyd算法检测图7.2中两个赋权有向图是否有负回路. 当无负回路时,输出图中任意两点之间的最短路径及其距离;当有负回路时,输出一条负回路.

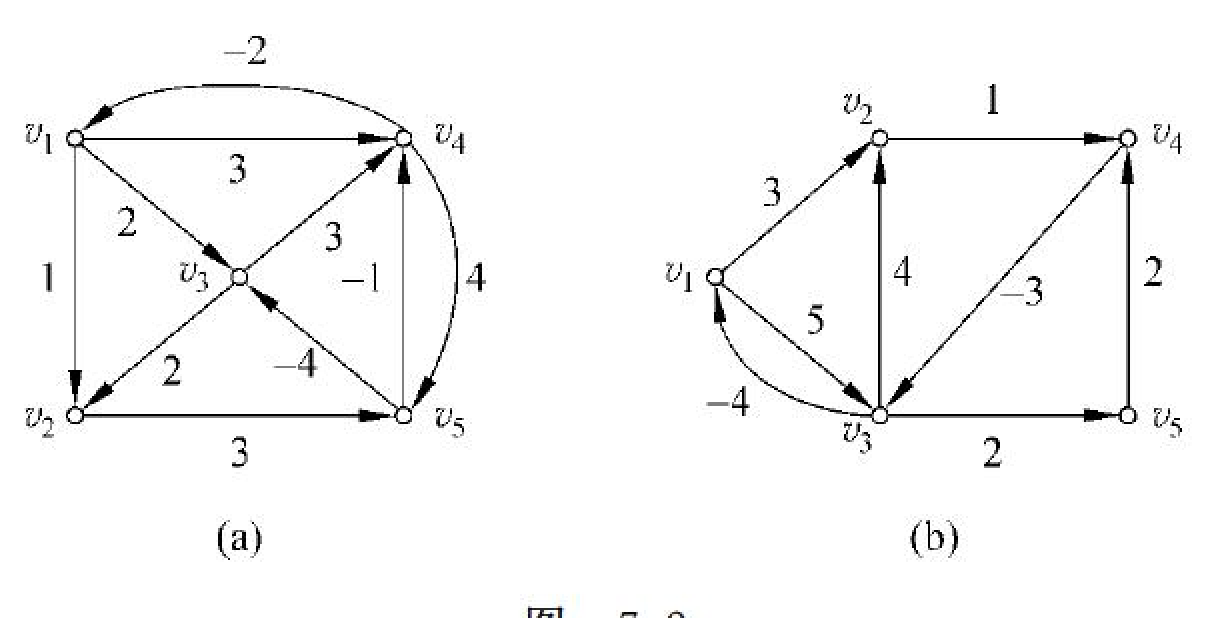


图 7.2

7.12 设赋权有向图 $D=<V,E,w>$ 无负回路,$n=|V|$,$d^{(k)}(i)$ 为 D 中 v_1 到 v_i 边数不超过 k 的最短路径的权,$2\leqslant i\leqslant n$,$k\geqslant 1$.

(1) 给出 $d^{(k)}(i)$ 的递推公式.

(2) 利用 $d^{(k)}(i)$ 的递推公式设计一个算法,求 D 中 v_1 到其他各点的最短路径,并分析算法的运行时间.

7.13 用在习题 7.12 中设计的算法计算图 7.2(a)中 v_1 到其他各点的最短路径.

7.14 (例 7.4 的继续)容量-费用网络 N 及流量 $v_0=8$ 的可行流 f_0 如图 7.3 所示，其中每一条边旁边的两个数依次是容量和费用. 以 f_0 为初始可行流用负回路算法，求 N 的流量 $v_0=8$ 的最小费用流.

7.15 用最短路径算法求图 7.3 中容量-费用网络 N 的流量 $v_0=8$ 的最小费用流(不使用图 7.3 中的 f_0).

7.16 设容量网络 $N=\langle V,E,c,s,t\rangle$，证明：存在最大流 f 使得，对于所有进入 s 的边和离开 t 的边 e，$f(e)=0$(因此，可以删去 N 中所有进入 s 的边和离开 t 的边. 也就是说，可以假设 s 的入度和 t 的出度为 0).

7.17 给定容量-费用网络，如何求下述最大流：

(1) 最小费用最大流.

(2) 费用不超过给定值的最大流.

图 7.3

7.18 某公司有 3 个工厂、4 个销售中心，各厂的产量、销售中心的销量以及它们之间的单位运费如表 7.1 所示. 试制订调运方案使得总运费最小.

表 7.1

工厂 \ 销售中心	销售中心Ⅰ	销售中心Ⅱ	销售中心Ⅲ	销售中心Ⅳ	产量 a_i
工厂 A	3	2	7	6	5
工厂 B	7	5	2	3	6
工厂 C	2	5	4	5	2.5
销量 b_j	6	4	2	1.5	13.5

7.19 某公司有 3 个生产基地、4 个用户，基地的产量、用户的需求量以及基地到用户的单位运费如表 7.2 所示. 试制订调运方案使得总运费最小.

表 7.2

基地 \ 用户	用户 B_1	用户 B_2	用户 B_3	用户 B_4	产量 a_i
基地 A_1	3	12	3	4	8
基地 A_2	11	2	5	9	5
基地 A_3	6	7	1	5	9
需求量 b_j	4	3	5	6	18 \ 22

7.20 某公司有 2 个工厂、2 个仓库和 4 个销售点. 工厂生产的产品均运往仓库，销售点从仓库提货. 假设仓库足够大，对存放的数量不构成限制. 图 7.4 给出产品的转运关系，边旁的数是单位运费. 试制订调运方案使得总运费最小.

7.21 某企业和用户签订了为期一年的合同，合同规定每季度末交货. 表 7.3 列出企业每季度的生产能力、交货量以及生产成本. 每台设备每季度的库存费 0.1 万元. 试制订生产计划使得总成本最低.

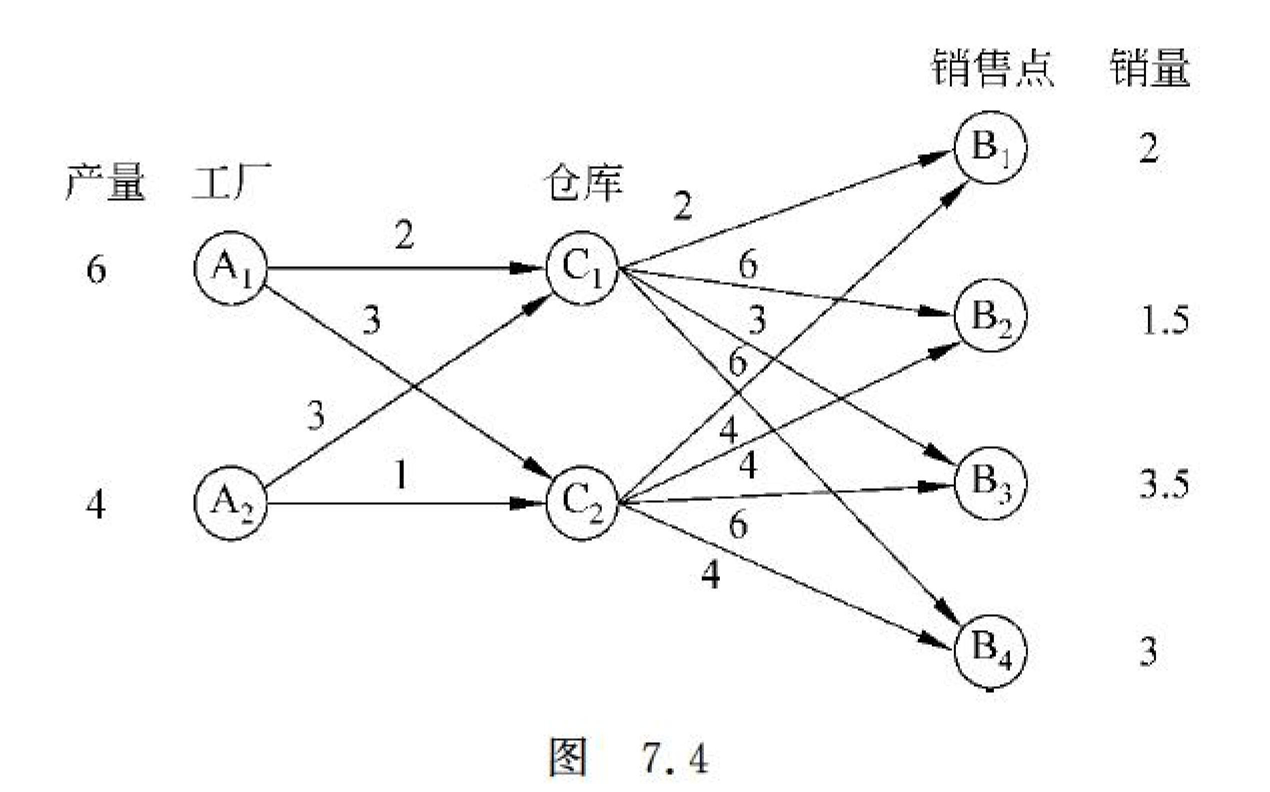


图 7.4

表 7.3

季度	生产能力/台	交货量/台	生产成本/(万元/台)
1	25	15	12.0
2	35	20	11.0
3	30	25	11.5
4	20	20	12.5

(1) 写出问题的最小费用流模型.

(2) 写出问题的运输问题模型并求解.

7.22 某社区有 5 个工作岗位,每个岗位需要一个人. 现接到 5 位待业者的申请,A 申请岗位 1、2 或 3,B 申请岗位 1 或 4,C 和 D 申请 4 或 5,E 申请岗位 5. 如何安排才能使尽可能多的申请者就业?

7.23 赋权完全二部图的权函数由图 7.5 所示的矩阵给出,求解这个指派问题.

$$\begin{bmatrix} 8 & 7 & 8 & 4 & 4 \\ 2 & 4 & 7 & 2 & 9 \\ 8 & 3 & 3 & 1 & 8 \\ 5 & 9 & 9 & 5 & 6 \\ 4 & 1 & 7 & 9 & 2 \end{bmatrix}$$

图 7.5

7.24 有 5 项工程由 3 家建筑公司承建. 建筑公司呈报的工程造价(千万元)如表 7.4 所示. 每家建筑公司最多承建 2 项工程. 如何安排才能使总造价最小?

表 7.4

	工程 B_1	工程 B_2	工程 B_3	工程 B_4	工程 B_5
公司 A_1	10	15	9	8	10
公司 A_2	9	18	6	10	8
公司 A_3	6	14	8	8	6

7.25 如果在算法 7.8(二部图最小权完美匹配的匈牙利算法)中直接使用公式(7.20)计算 θ,对算法的时间复杂度有什么影响?

7.26 设二部图 $G=\langle A,B,E\rangle$,函数 b: $A\cup B\to \mathbf{Z}^+$,$M\subseteq E$,其中 $\mathbf{Z}^+$ 是正整数集. 如果 $\forall v\in A\cup B$,v 恰好与 M 中的 $b(v)$ 条边关联,则称 M 是 G 的 ***b*-匹配**. b-匹配问题就是给定二部图 $G=\langle A,B,E\rangle$ 和函数 b:$A\cup B\to\mathbf{Z}^+$,求 G 的 b-匹配. 试设计一个 b-匹配问题的算法并分析算法的时间复杂度.

7.27 二部图的**瓶颈匹配**问题:给定赋权二部图 $G=\langle A,B,E,w\rangle$,求 G 的最大权最

小的完美匹配 M，即求完美匹配 M 是使得 $\max\{w(e)|e\in M\}$ 最小. 设计一个二部图瓶颈匹配问题的算法并分析算法的时间复杂度.

7.28 设无向图 $G=\langle V,E\rangle$，$V'\subseteq V$. 如果每一条边都有一个端点属于 V'，则称 V' 是一个**顶点覆盖**. 顶点数最少的顶点覆盖称作**最小顶点覆盖**.

(1) 设二部图 $G=\langle A,B,E\rangle$，$M\subseteq E$ 是一个匹配，$V\subseteq A\cup B$ 是一个顶点覆盖. 证明：$|M|\leqslant|V|$.

(2) 设二部图 $G=\langle A,B,E\rangle$，M 是最大匹配，用匈牙利算法计算中，A 中未标号的顶点集为 A_1，B 中已标号的顶点集为 B_1，$V=A_1\cup B_1$. 证明：V 是一个顶点覆盖且 $|V|=|M|$.

(3) 设计一个求二部图最小顶点覆盖的算法.

7.3 习题解答与分析

7.1

$$\begin{aligned}
v(f) &= \sum_{\langle s,j\rangle\in E} f(s,j) - \sum_{\langle j,s\rangle\in E} f(j,s) \\
&= \sum_{\langle s,j\rangle\in E} f(s,j) - \sum_{\langle j,s\rangle\in E} f(j,s) + \sum_{j\neq s,t}\Big\{\sum_{\langle j,i\rangle\in E} f(j,i) - \sum_{\langle i,j\rangle\in E} f(i,j)\Big\} \\
&= \sum_{\substack{\langle s,j\rangle\in E\\ j\neq t}} f(s,j) + f(s,t) - \sum_{\substack{\langle j,s\rangle\in E\\ j\neq t}} f(j,s) - f(t,s) \\
&\quad + \sum_{\substack{\langle j,i\rangle\in E\\ i\neq s\\ j\neq s,t}} f(j,i) + \sum_{\substack{\langle j,s\rangle\in E\\ j\neq t}} f(j,s) - \sum_{\substack{\langle i,j\rangle\in E\\ i\neq s\\ j\neq s,t}} f(i,j) - \sum_{\substack{\langle s,j\rangle\in E\\ j\neq t}} f(s,j) \\
&= f(s,t) - f(t,s) \\
&\quad + \sum_{\substack{\langle j,i\rangle\in E\\ i\neq s,t\\ j\neq s,t}} f(j,i) + \sum_{\substack{\langle j,t\rangle\in E\\ j\neq s}} f(j,t) - \sum_{\substack{\langle i,j\rangle\in E\\ i\neq s,t\\ j\neq s,t}} f(i,j) - \sum_{\substack{\langle t,j\rangle\in E\\ j\neq s}} f(t,j) \\
&= \sum_{\langle j,t\rangle\in E} f(j,t) - \sum_{\langle t,j\rangle\in E} f(t,j)
\end{aligned}$$

7.2 计算过程如图 7.6 所示. 顶点旁括号内是它的标号. 每个图的左边给出检查顶点的顺序，括号内是检查这个顶点时得到标号的顶点. 粗边是找到的增广链. 最大流 $f_{\max}$ 如图 7.6(d)所示. 在图 7.6(d)中，虚线将顶点集划分成两部分，给出最小割集，$X=\{s,1,2,3,4\}$，$V-X=\{5,t\}$，最小割集 $(X,V-X)=\{\langle 2,5\rangle,\langle 3,5\rangle,\langle 4,t\rangle\}$.

$$v(f_{\max}) = 8+10 = 18 = 3+5+10 = c(X,V-X)$$

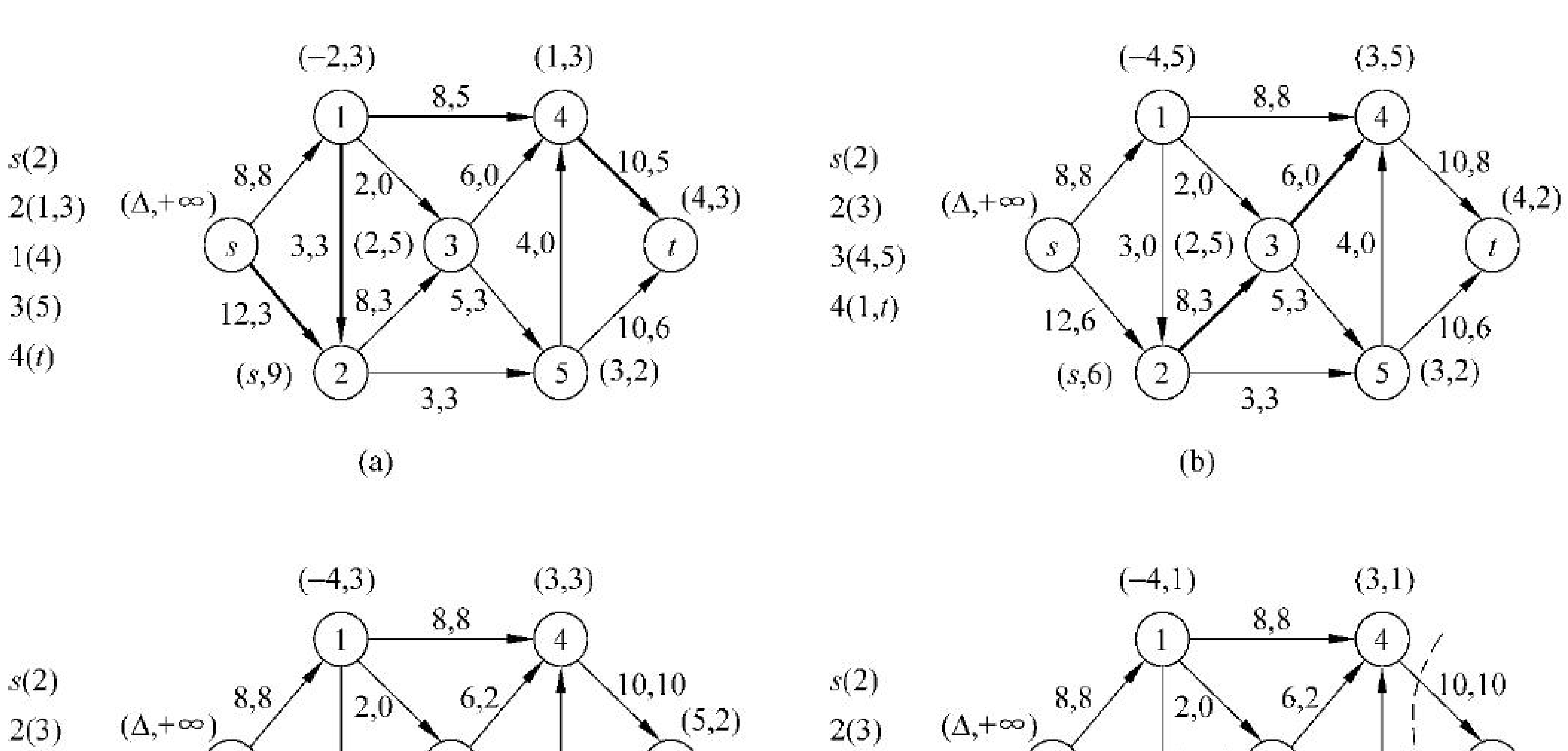
7.3 计算过程如图 7.7 所示. 图 7.7(b)和(h)中顶点旁方括号内的数字是顶点的层次，图 7.7(c)、(d)和(e)中 * 表示该顶点的流通量最小，δ 是本次在该边安排的流量值. 图 7.7(h)的 $N(f)$ 中不存在 s-t 路径，故 f 是最大流. 在图 7.7(d)中，顶点 3 的流通量也是最小的. 如果从顶点 3 而不是从顶点 1 开始安排流量，则会得到另一个最大流.

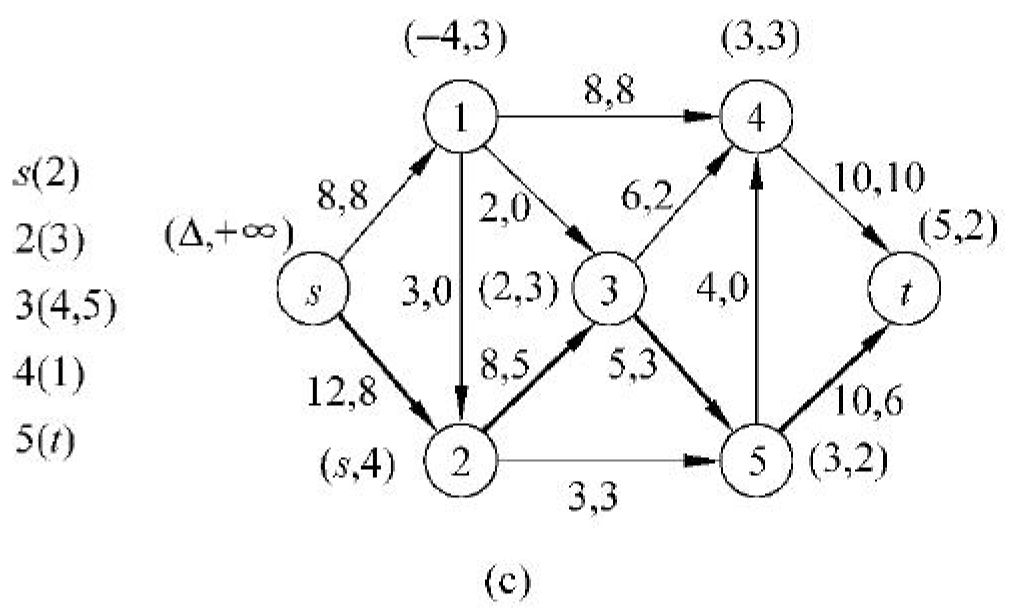
7.4 设 $N=\langle V,E,c,s,t\rangle$，$N(f)=\langle V,E(f),ac,s,t\rangle$，其中

$$E(f) = E^+(f)\cup E^-(f)$$
$$E^+(f) = \{\langle i,j\rangle \mid \langle i,j\rangle\in E \wedge f(i,j)=0\}$$
$$E^-(f) = \{\langle i,j\rangle \mid \langle j,i\rangle\in E \wedge f(j,i)=1\}$$

对于 $\forall\langle i,j\rangle\in E(f)$，若 $\langle i,j\rangle\in E^+(f)$，则 $f(i,j)=0$，$ac(i,j)=c(i,j)-f(i,j)=1$；若 $\langle i,j\rangle\in E^-(f)$，则 $ac(i,j)=f(j,i)=1$. 得证 $N(f)$ 所有边的容量为 1.


116


s(2)
2(1,3)
1(4)
3(5)
4(t)
(−2,3)
(1,3)
8,5
8,8
2,0
6,0
10,5
(4,3)
(Δ,+∞)
3,3
(2,5)
4,0
12,3
8,3
5,3
10,6
(s,9)
(3,2)
3,3
(a)
s(2)
2(3)
3(4,5)
4(1,t)
(−4,5)
(3,5)
8,8
8,8
2,0
6,0
10,8
(4,2)
(Δ,+∞)
3,0
(2,5)
4,0
12,6
8,3
5,3
10,6
(s,6)
(3,2)
3,3
(b)


s(2)
2(3)
3(4,5)
4(1)
5(t)
(−4,3)
(3,3)
8,8
8,8
2,0
6,2
10,10
(5,2)
(Δ,+∞)
3,0
(2,3)
4,0
12,8
8,5
5,3
10,6
(s,4)
(3,2)
3,3
(c)

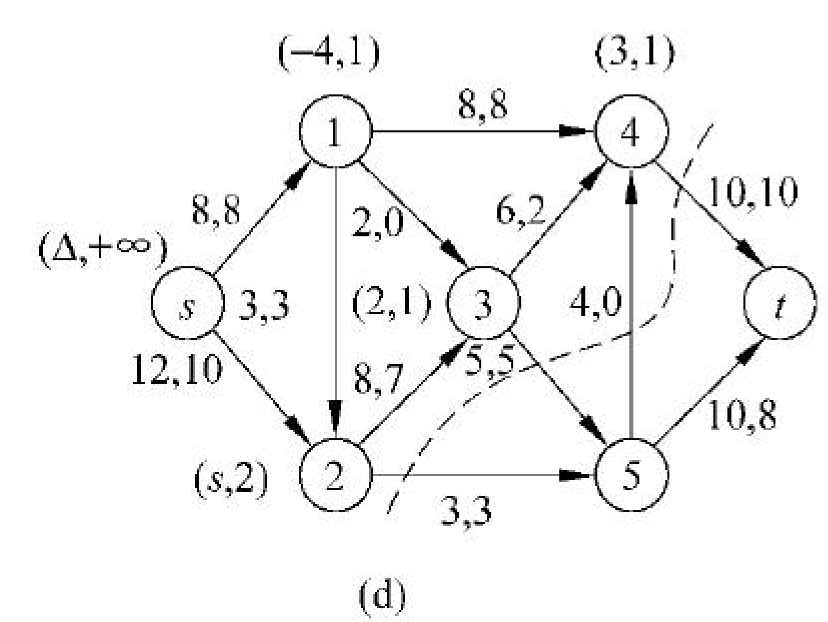

s(2)
2(3)
3(4)
4(1)
1( )
(−4,1)
(3,1)
8,8
8,8
2,0
6,2
10,10
(Δ,+∞)
3,3
(2,1)
4,0
12,10
8,7
5,5
10,8
(s,2)
3,3
(d)

图 7.6

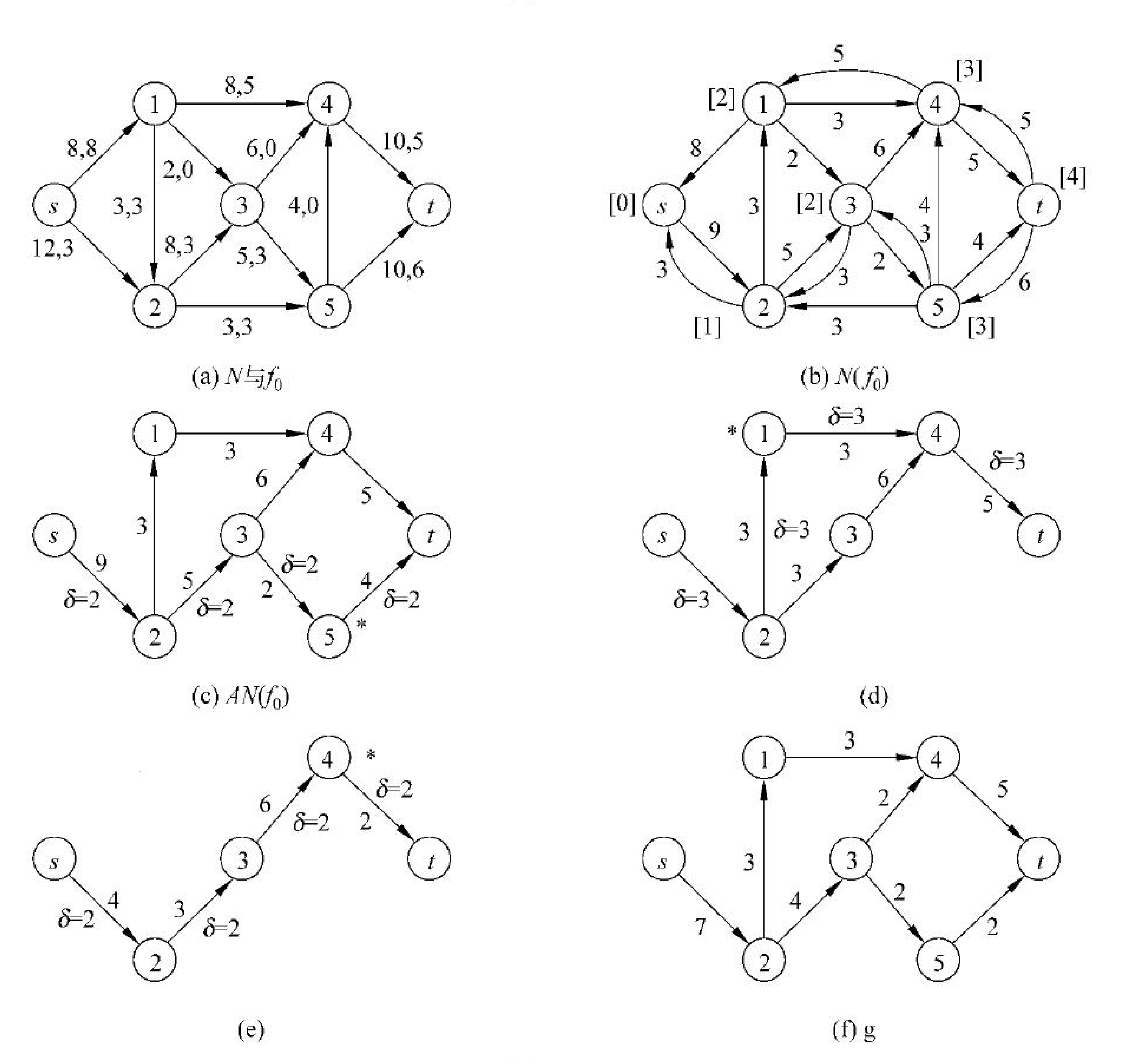
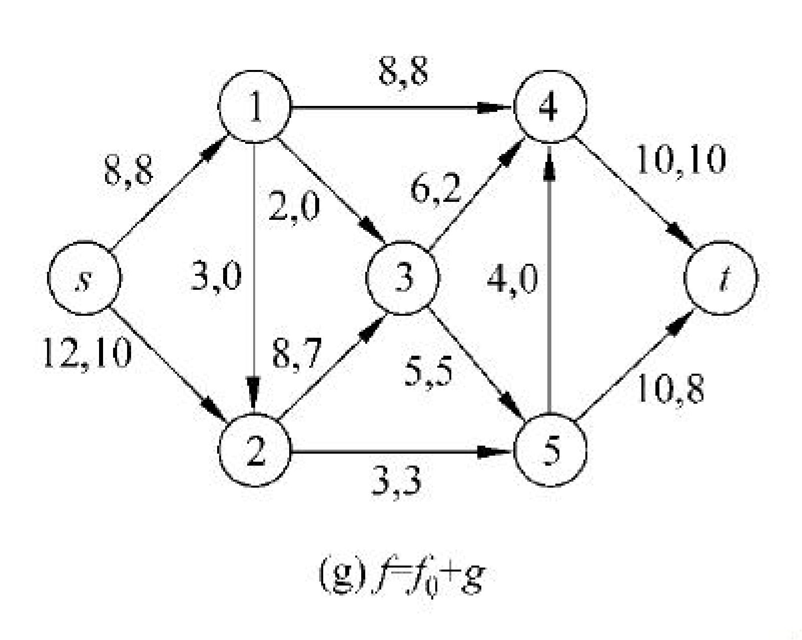
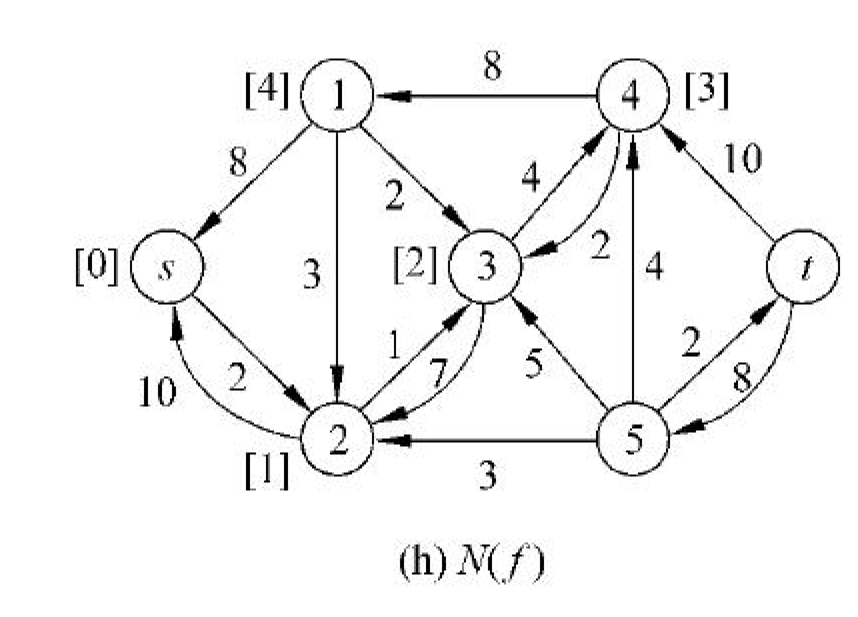

(a) N与f_0
(b) N(f_0)
(c) AN(f_0)
(d)
(e)
(f) g

图 7.7



(g) $f=f_0+g$



(h) $N(f)$

图 7.7(续)

对于 $\forall i \in V-\{s,t\}$,如果 i 在 N 中的入度为 1,设 $<p,i>,<i,q_1>,\cdots,<i,q_r>\in E$,有以下两种可能:

(1) $f(p,i)=1$,则 $f(i,q_1),\cdots,f(i,q_r)$ 中有一个等于 1,其余都等于 0. 不妨设,$f(i,q_1)=1,f(i,q_2)=\cdots=f(i,q_r)=0$. 于是,$<q_1,i>,<i,p>,<i,q_2>,\cdots,<i,q_r>\in E$,$i$ 在 $N(f)$ 中的入度为 1.

(2) $f(p,i)=0$,则 $f(i,q_1),\cdots,f(i,q_r)$ 全等于 0. 于是,$<p,i>,<i,q_1>,\cdots,<i,q_r>\in E$,$i$ 在 $N(f)$ 中的入度也为 1.

类似可证,如果 i 在 N 中的出度为 1,则 i 在 $N(f)$ 中的出度也为 1. 得证 $N(f)$ 是简单容量网络.

7.5 方法一:设容量网络 $N=<V,E,c,s,t>$,给定流量 v_0. 求 N 上流量 v_0 的可行流可转化成求 N' 上的最大流. 引入一个新的发点 s_0,$c(s_0,s)=v_0$,而原来的发点 s 成为中间点. 即 $N'=<V\cup\{s_0\},E\cup\{<s_0,s>\},c',s_0,t\}$,其中

$$c'(i,j)=\begin{cases}v_0, & i=s_0 \wedge j=s \\ c(i,j), & <i,j>\in E\end{cases}$$

算法如下:

1. 构造 N'.

2. 求 N' 的最大流 $f_{\max}$.

3. 若 $v(f_{\max})=v_0$,则在 N 上 $f_{\max}$ 即为流量 v_0 的可行流. 若 $v(f_{\max})<v_0$,则 N 上不存在流量 v_0 的可行流.

方法二:修改最大流算法中调整流量的值使得可行流的流量不超过 v_0. 如对于 FF 算法,设当前的可行流为 f,$v(f)<v_0$,找到增广链 P,P 上流量的调整量为 δ(即 P 上前向边的容量与流量之差以及后向边的流量中的最小值). 现在把 P 上流量的调整量修改为 $\min(v_0-v(f),\delta)$. 计算直到 $v(f)=v_0$ 停止. 如果已得到最大流 f 且 $v(f)<v_0$,则 N 上不存在流量 v_0 的可行流.

7.6 设多发点多收点容量网络 $N=<V,E,c,S,T>$,其中 $S\subset V$,$T\subset V$,且 $S\cap T=\varnothing$. 如下构造容量网络 $N'=<V',E',c',s_0,t_0>$,其中

$$V'=V\cup\{s_0,t_0\}$$

$$E'=E\cup\{<s_0,s>\mid s\in S\}\cup\{<t,t_0>\mid t\in T\}$$

$$c'(i,j)=\begin{cases}c_0 & (i=s_0 \wedge j\in S)\vee(i\in T\wedge j=t_0) \\ c(i,j) & <i,j>\in E\end{cases}$$

这里 $s_0, t_0 V$, $c_0 = \sum_{s\in S}\sum_{<s,j>\in E} c(s,j)$.

设 f 是 N 的一个可行流,把它扩张到 N'上. 令

$$f'(i,j)=\begin{cases}\sum_{<s,k>\in E} f(s,k)-\sum_{<k,s>\in E} f(k,s), & i=s_0 \wedge j=s\in S\\ \sum_{<k,t>\in E} f(k,t)-\sum_{<t,k>\in E} f(t,k), & j=t_0 \wedge i=t\in T\\ f(i,j), & <i,j>\in E\end{cases}$$

不难验证 f'在 S 和 T 中所有的顶点都满足平衡条件,在新添加的边上满足容量限制,从而 f'是 N'的一个可行流,且 $v(f')=v(f)$.

反之,设 f'是 N'上的一个可行流,把 f'在 N 上的限制记作 f,f 是 N 上的可行流,且 $v(f)=v(f')$. 从而 f 是 N 的最大流当且仅当 f'是 N'的最大流. 这就把求 N 的最大流转化为求 N'的最大流.

这种把一个问题转化为另一个问题的方法称作**归约**. 习题 7.5 中的方法一用的也是归约. 在实际应用中,如果能够把要解决的问题归约为一个已知的模型,那么这个问题就解决了. 如这里把多发点、多收点容量网络的最大流问题归约为标准的最大流问题. 通过求 N'的最大流,就可以得到 N 的最大流. 当然,为了保证算法的效率,归约所用的变换不能过于复杂,不能使计算时间增加太多. 归约不但在实际中有广泛的应用,而且在理论上是一个十分重要的概念(第 9 章将会详细介绍).

7.7 把每一个中间顶点 v 替换成两个顶点 $v(1)$,$v(2)$ 和一条边 $<v(1),v(2)>$,$c(v(1),v(2))=c(v)$. 每一条边 $<u,v>$ 替换成 $<u(2),v(1)>$,$c(u(2),v(1))=c(u,v)$. 这里约定,$s(1)=s(2)=s$,$t(1)=t(2)=t$. 即如下构造容量网络 $N'=<V',E',c',s,t>$,其中:

$$V'=\{s,t\}\cup\{v(1),v(2)\mid v\in V-\{s,t\}\}$$

$$E'=\{<u(2),v(1)>\mid <u,v>\in E\}\cup\{<v(1),v(2)>\mid v\in V-\{s,t\}\}$$

$$c'(e)=\begin{cases}c(u,v), & e=<u(2),v(1)>,\\ c(v), & e=<v(1),v(2)>,\end{cases}\quad \forall e\in E'$$

不难把 N 上的可行流转换成 N'上的可行流,且其流量相同,反之亦然.

7.8 (1) 设 s-t 距离为 d,V_i为与 s 距离 i 的顶点集,$0\leqslant i\leqslant d$,则

$$v^*\leqslant c(V_i,V_{i+1})\leqslant |V_i|\times|V_{i+1}|,\quad 0\leqslant i\leqslant d-1$$

于是,$|V_i|$ 和 $|V_{i+1}|$ 中必有一个不小于 $\sqrt{v^*}$,$d+1$ 个 $|V_i|$ 中至少有 $d/2$ 个不小于 $\sqrt{v^*}$,从而 $n\geqslant\frac{d}{2}\sqrt{v^*}$,得证 $d\leqslant 2n/\sqrt{v^*}$.

(2) 由于 N 具有单位容量,Dinic 算法在每个阶段对每一条边至多处理一次,故每个阶段的所需时间为 $O(m)$.

要证明阶段数为 $O(n^{2/3})$. 如果 $v^*\leqslant n^{2/3}$,则结论成立. 假设 $v^*>n^{2/3}$,现考察可行流的流量第一次超过 $v^*-n^{2/3}$ 的阶段. 在这个阶段之后最多还有 $n^{2/3}$ 个阶段. 现在考虑在这个阶段之前有多少个阶段. 设这个阶段开始时的可行流为 f,$v(f)\leqslant v^*-n^{2/3}$,于是 $N(f)$的最大流量不小于 $n^{2/3}$. 不难验证 $N(f)$具有单位容量,根据本题(1),在 $N(f)$中 s-t 距离小于等于 $2n/\sqrt{n^{2/3}}=2n^{2/3}$. 而每个阶段的 s-t 距离都大于前一个阶段的 s-t 距离,故在这个阶段之前最多有 $2n^{2/3}$ 个阶段,总共有 $3n^{2/3}+1$ 个阶段.

7.9 把 D 看作一个容量网络 N，边的容量都为 1. 设 f 是 N 的一个可行流，则所有 $f(e)=1$ 的边构成边不相交的 s-t 路径，且路径数等于 $v(f)$. 对 $v(f)$ 作归纳证明，当 $v(f)=1$ 时，结论显然成立. 假设当 $v(f)=k\geqslant 1$ 时结论成立，现考虑 $v(f)=k+1$. 从 s 开始沿着 $f(e)=1$ 的边走到 t，得到一条 s-t 路径 L. 把 f 在 L 上的值改为 0，记作 f'. f' 也是 N 上的可行流且 $v(f')=k$. 根据归纳假设，所有 $f'(e)=1$ 的边构成 k 条边不相交的 s-t 路径. 这些路径与 L 没有共同的边，故当 $v(f)=k+1$ 时结论也成立.

反之，设 E' 是 D 中边不相交的 s-t 路径的边集合. $\forall e\in E'$，令 $f(e)=1$；$\forall e\in E-E'$，令 $f(e)=0$. 可对路径数归纳证明：f 是 N 上的一个可行流，且 $v(f)$ 等于 E' 中的路径数. 于是，D 上条数最多的边不相交的 s-t 路径恰好对应 N 上的最大流. 这样就把求 D 中尽可能多的边不相交的 s-t 路径转化为 N 的最大流问题.

算法描述如下：

步骤 1：构造容量网络 $N=\langle V,E,c,s,t\rangle$，其中所有边 e 的容量 $c(e)=1$.

步骤 2：求 N 的最大流 $f_{\max}$.

步骤 3：用标号法把 $f_{\max}$ 转换成 D 中的 s-t 路径.

```
1. for v∈V do l(v)←∅              //l(v)是所有路径中关联到 v 的顶点集合
2. for <u,v>∈E do
2.1     if f_max(u,v)=1 then l(v)←l(v)∪{u}
3. Q←{t}, P←∅                     //P 存放所有边不相交的 s-t 路径
4. while Q≠∅ do
4.1     任取 L∈Q, v←L 的第一个顶点, Q←Q-{L}
4.2     if v≠s then
4.3         for u∈l(v) do Q←Q∪{uL}
4.4     else P←P∪{L}
5. return P
```

分析算法的运行时间. 步骤 1 只需要生成 c，可用 $O(m)$ 步完成. 由于所有边不相交的 s-t 路径的长度之和不超过 m，故步骤 3 可在 $O(m)$ 步完成. 步骤 2 若采用 FF 算法，由于所有容量为 1，故得到的最大流在每一条边上的值是 0 或 1. 最大流的流量不超过 s 的出度，必小于 n. 而每个阶段流量至少增加 1，故阶段数小于 n. 每个阶段的运行时间为 $O(m)$，步骤 2 的运行时间为 $O(nm)$. 算法的运行时间为 $O(nm)$. 步骤 2 若采用 Dinic 算法，由于 N 具有单位容量，根据习题 7.8，步骤 2 的运行时间为 $O(n^{2/3}m)$. 算法的运行时间为 $O(n^{2/3}m)$.

7.10 把 D 看作一个带顶点容量的容量网络，所有的边和所有的顶点的容量都为 1. 可以类似习题 7.9 证明，能够把求 D 中尽可能多的顶点不相交的 s-t 路径转化为这个带顶点容量的容量网络的最大流问题. 根据习题 7.7，这个带顶点容量的容量网络的最大流问题又可转化为标准的最大流问题. 后者的容量网络为 $N=\langle V',E',c,s,t\rangle$，其中

$$V' = \{s,t\} \cup \{v(1),v(2) \mid v \in V-\{s,t\}\}$$

$$E' = \{\langle u(2),v(1)\rangle \mid \langle u,v\rangle \in E\} \cup \{\langle v(1),v(2)\rangle \mid v \in V-\{s,t\}\}$$

$$c(e) = 1 \quad e \in E'$$

这里约定，$s(1)=s(2)=s$，$t(1)=t(2)=t$.

算法如下：

步骤1:构造 $N=\langle V',E',c,s,t\rangle$.

步骤2:求 N 的最大流 f_{max}.

步骤3:根据 f_{max} 构造 D 的顶点不相交的 s-t 路径集合(类似习题7.8中的算法步骤3).

算法分析:记 $n=|V|$,$m=|E|$,则 $|V'|=2n-2$,$|E'|=m+n-2$. 注意到,N 是简单容量网络. 类似习题7.9可证,若在步骤2采用FF算法,算法的时间复杂度为 $O(nm)$. 若在步骤2采用Dinic算法,算法的时间复杂度为 $O(n^{1/2}m)$.

7.11 图(a)中,沿用教材中的符号,$d^{(k)}(i,j)$ 是 v_i 到 v_j 之间经过下标不超过 k 的最短路径的长度,$h^{(k)}(i,j)$ 是这条路径中 v_i 的下一个顶点的下标. 计算过程如下

$$d^{(0)}=\begin{bmatrix}0&1&2&3&+\infty\\+\infty&0&+\infty&+\infty&3\\+\infty&2&0&3&+\infty\\-2&+\infty&+\infty&0&4\\+\infty&+\infty&-4&-1&0\end{bmatrix}\qquad h^{(0)}=\begin{bmatrix}1&2&3&4&0\\0&2&0&0&5\\0&2&3&4&0\\1&0&0&4&5\\0&0&3&4&5\end{bmatrix}$$

$$d^{(1)}=\begin{bmatrix}0&1&2&3&+\infty\\+\infty&0&+\infty&+\infty&3\\+\infty&2&0&3&+\infty\\-2&-1&0&0&4\\+\infty&+\infty&-4&-1&0\end{bmatrix}\qquad h^{(1)}=\begin{bmatrix}1&2&3&4&0\\0&2&0&0&5\\0&2&3&4&0\\1&1&1&4&5\\0&0&3&4&5\end{bmatrix}$$

$$d^{(2)}=\begin{bmatrix}0&1&2&3&4\\+\infty&0&+\infty&+\infty&3\\+\infty&2&0&3&5\\-2&-1&0&0&2\\+\infty&+\infty&-4&-1&0\end{bmatrix}\qquad h^{(2)}=\begin{bmatrix}1&2&3&4&2\\0&2&0&0&5\\0&2&3&4&2\\1&1&1&4&1\\0&0&3&4&5\end{bmatrix}$$

$$d^{(3)}=\begin{bmatrix}0&1&2&3&4\\+\infty&0&+\infty&+\infty&3\\+\infty&2&0&3&5\\-2&-1&0&0&2\\+\infty&-2&-4&-1&0\end{bmatrix}\qquad h^{(3)}=\begin{bmatrix}1&2&3&4&2\\0&2&0&0&5\\0&2&3&4&2\\1&1&1&4&1\\0&3&3&4&5\end{bmatrix}$$

$$d^{(4)}=\begin{bmatrix}0&1&2&3&4\\+\infty&0&+\infty&+\infty&3\\1&2&0&3&5\\-2&-1&0&0&2\\-3&-2&-4&-1&0\end{bmatrix}\qquad h^{(4)}=\begin{bmatrix}1&2&3&4&2\\0&2&0&0&5\\4&2&3&4&2\\1&1&1&4&1\\4&3&3&4&5\end{bmatrix}$$

$$d^{(5)}=\begin{bmatrix}0&1&0&3&4\\0&0&-1&2&3\\1&2&0&3&5\\-2&-1&-2&0&2\\-3&-2&-4&-1&0\end{bmatrix}\qquad h^{(5)}=\begin{bmatrix}1&2&2&4&2\\5&2&5&5&5\\4&2&3&4&2\\1&1&1&4&1\\4&3&3&4&5\end{bmatrix}$$

$d^{(5)}$的对角线元素均为0,图中没有负回路. $d^{(5)}$给出任意两点之间的距离,根据$h^{(5)}$可以查找到任意两点之间的最短路径. 例如,v_1到v_3的距离等于$d^{(5)}(1,3)=0$. 由$h^{(5)}(1,3)=2$, $h^{(5)}(2,3)=5$, $h^{(5)}(5,3)=3$, v_1到v_3的最短路径是$v_1v_2v_5v_3$. 又如,v_5到v_2的距离等于$d^{(5)}(5,2)=-2$. 由$h^{(5)}(5,2)=3$, $h^{(5)}(3,2)=2$, v_5到v_2的最短路径是$v_5v_3v_2$.

图(b)计算如下:

$$d^{(0)}=\begin{bmatrix} 0 & 3 & 5 & +\infty & +\infty \\ +\infty & 0 & +\infty & 1 & +\infty \\ -4 & 4 & 0 & +\infty & 2 \\ +\infty & +\infty & -3 & 0 & +\infty \\ +\infty & +\infty & +\infty & 2 & 0 \end{bmatrix} \qquad h^{(0)}=\begin{bmatrix} 1 & 2 & 3 & 0 & 0 \\ 0 & 2 & 0 & 4 & 0 \\ 1 & 2 & 3 & 0 & 5 \\ 0 & 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 4 & 5 \end{bmatrix}$$

$$d^{(1)}=\begin{bmatrix} 0 & 3 & 5 & +\infty & +\infty \\ +\infty & 0 & +\infty & 1 & +\infty \\ -4 & -1 & 0 & +\infty & 2 \\ +\infty & +\infty & -3 & 0 & +\infty \\ +\infty & +\infty & +\infty & 2 & 0 \end{bmatrix} \qquad h^{(1)}=\begin{bmatrix} 1 & 2 & 3 & 0 & 0 \\ 0 & 2 & 0 & 4 & 0 \\ 1 & 1 & 3 & 0 & 5 \\ 0 & 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 4 & 5 \end{bmatrix}$$

$$d^{(2)}=\begin{bmatrix} 0 & 3 & 5 & 4 & +\infty \\ +\infty & 0 & +\infty & 1 & +\infty \\ -4 & -1 & 0 & 0 & 2 \\ +\infty & +\infty & -3 & 0 & +\infty \\ +\infty & +\infty & +\infty & 2 & 0 \end{bmatrix} \qquad h^{(2)}=\begin{bmatrix} 1 & 2 & 3 & 2 & 0 \\ 0 & 2 & 0 & 4 & 0 \\ 1 & 1 & 3 & 1 & 5 \\ 0 & 0 & 3 & 4 & 0 \\ 0 & 0 & 0 & 4 & 5 \end{bmatrix}$$

$$d^{(3)}=\begin{bmatrix} 0 & 3 & 5 & 4 & 7 \\ +\infty & 0 & +\infty & 1 & +\infty \\ -4 & -1 & 0 & 0 & 2 \\ -7 & -4 & -3 & -3 & \sim \\ \sim & \sim & \sim & \sim & \sim \end{bmatrix} \qquad h^{(3)}=\begin{bmatrix} 1 & 2 & 3 & 2 & 3 \\ 0 & 2 & 0 & 4 & 0 \\ 1 & 1 & 3 & 1 & 5 \\ 3 & 3 & 3 & 3 & \sim \\ \sim & \sim & \sim & \sim & \sim \end{bmatrix}$$

至此,找到图中过v_4的一条负回路$v_4v_3v_1v_2v_4$,长为-3.

7.12 (1) 设P是v_1到v_i边数不超过k的最短路径,如果P的边数小于k,则P的权$w(P)=d^{(k-1)}(i)$. 如果P的边数等于k,那么P必由v_1经过$k-1$条边到某个v_j的最短路径P'与边$<v_j,v_i>$组成,并且是所有这种可能中权最小的. 从而,有下述递推公式:

$$d^{(0)}(1)=0,\quad d^{(0)}(i)=+\infty,\quad 2\leqslant i\leqslant n$$

$$d^{(k)}(i)=\min\{d^{(k-1)}(i),\ \min\{d^{(k-1)}(j)+w_{ji}\mid <v_j,v_i>\in E\}\},\quad 1\leqslant i\leqslant n,\ k\geqslant 1$$

因为D中无负回路,两点之间一定存在边数不超过$n-1$的最短路径,所以$d^{(n-1)}(i)$是v_1到v_i的距离,$1\leqslant i\leqslant n$. 递推公式只需计算到$k=n-1$.

(2) 利用上述递推公式不难设计出所需的动态规划算法. 为了保存最短路径的信息,设$h^{(k)}(i)$为v_1到v_i边数不超过k的最短路径中v_i的前一个顶点的下标. 记$j^{(k)}(i)$为使$d^{(k-1)}(j)+w_{ji}(<v_j,v_i>\in E)$取到最小值的$v_j$的下标.

递推公式如下:

$$h^{(0)}(1)=1,\quad h^{(0)}(i)=0,\quad 2\leqslant i\leqslant n$$

$$h^{(k)}(i)=\begin{cases}h^{(k-1)}(i), & d^{(k-1)}(i)\leqslant \min\limits_{<v_j,v_i>\in E}\{d^{(k-1)}(j)+w_{ji}\},\\ j^{(k)}(i), & \text{否则},\end{cases}\quad 1\leqslant i\leqslant n,\ 1\leqslant k\leqslant n-1$$

算法如下：

```
d(1)←1, h(1)←1
for i=2 to n do
    d(i)←+∞, h(i)←0
for k=1 to n−1 do
    for i=1 to n do
        for <v_j,v_i>∈E do
            if d(j)+w_ji<d(i) then
                d(i)←d(j)+w_ji, h(i)←j
return d, h
```

容易看出算法的运行时间为 $O(n^3)$.

这个算法是由 L. R. Ford 提出的,故称作 **Ford 算法**. 在最小费用流的最短路算法中用 Ford 算法求费用最小的增广链更好些.

7.13 为了方便,将图 7.2(a)重绘于图 7.8. 计算如下：

$$d^{(0)}=(0,+\infty,+\infty,+\infty,+\infty),\quad h^{(1)}=(1,0,0,0,0)$$
$$d^{(1)}=(0,1,2,3,+\infty),\quad h^{(1)}=(1,1,1,1,0)$$
$$d^{(2)}=(0,1,2,3,4),\quad h^{(2)}=(1,1,1,1,2)$$
$$d^{(3)}=(0,1,0,3,4),\quad h^{(3)}=(1,1,5,1,2)$$
$$d^{(4)}=(0,1,0,3,4),\quad h^{(4)}=(1,1,5,1,2)$$

v_1 到 v_2 的最短路径是 v_1v_2,权为 1.

v_1 到 v_3 的最短路径是 $v_1v_2v_5v_3$,权为 0.

v_1 到 v_4 的最短路径是 v_1v_4,权为 3.

v_1 到 v_5 的最短路径是 $v_1v_2v_5$,权为 4.

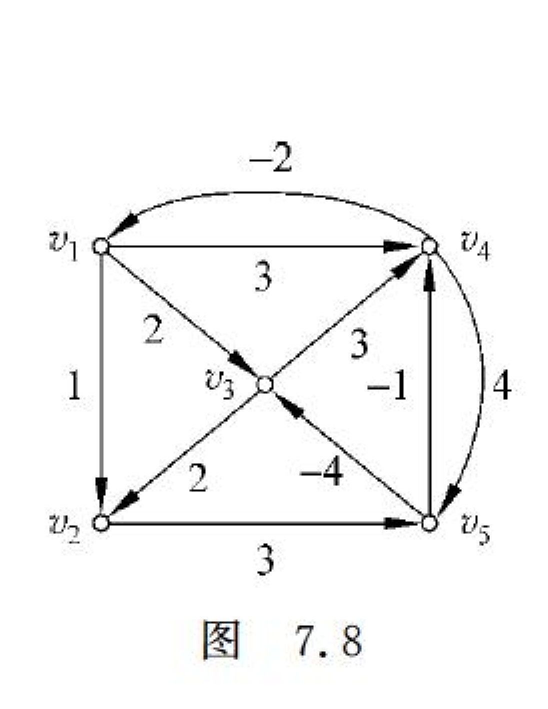


图 7.8

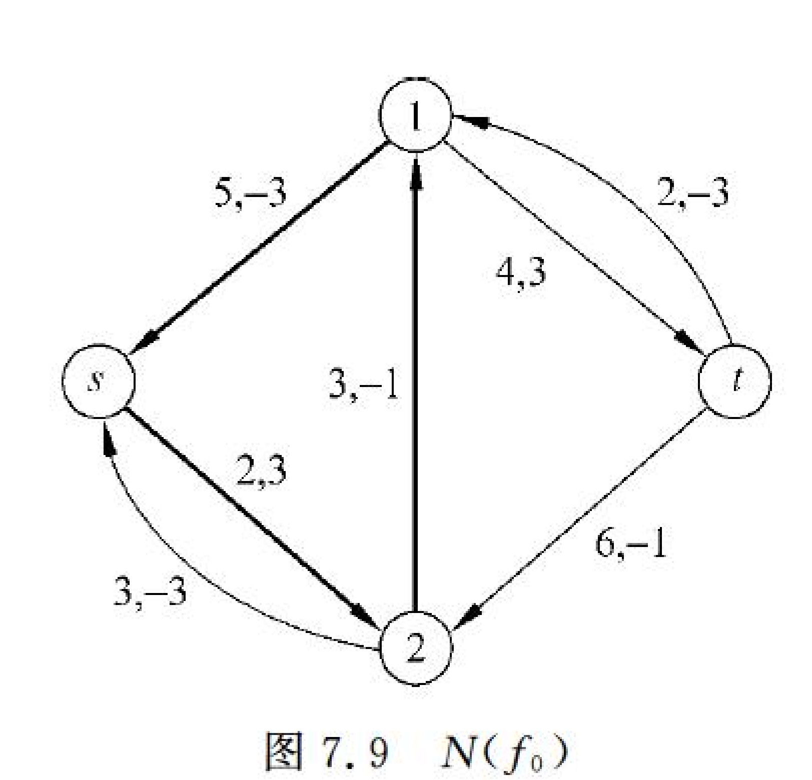


图 7.9 $N(f_0)$

7.14 $N(f_0)$如图 7.9 所示,用 Floyd 算法检查 $N(f_0)$中是否有关于 aw 的负回路,计算过程如下：

$$d^{(0)}=\begin{bmatrix}0&+\infty&3&+\infty\\-3&0&+\infty&3\\-3&-1&0&+\infty\\+\infty&-3&-1&0\end{bmatrix}\qquad h^{(0)}=\begin{bmatrix}s&0&2&0\\s&1&0&t\\s&1&2&0\\0&1&2&t\end{bmatrix}$$

$$d^{(s)}=\begin{bmatrix}0&+\infty&3&+\infty\\-3&0&0&3\\-3&-1&0&+\infty\\+\infty&-3&-1&0\end{bmatrix}\qquad h^{(s)}=\begin{bmatrix}s&0&2&0\\s&1&s&t\\s&1&2&0\\0&1&2&t\end{bmatrix}$$

$$d^{(1)}=\begin{bmatrix}0&+\infty&3&+\infty\\-3&0&0&3\\-4&-1&-1&\sim\\\sim&\sim&\sim&\sim\end{bmatrix}\qquad h^{(1)}=\begin{bmatrix}s&0&2&0\\s&1&s&t\\1&1&1&\sim\\\sim&\sim&\sim&\sim\end{bmatrix}$$

至此，$d^{(1)}(2,2)=-1$，过顶点 2 有一条负回路 C. 由 $h^{(1)}(2,2)=1$，$h^{(1)}(1,2)=s$，$h^{(1)}(s,2)=2$，$C=21s2$，如图 7.9 中粗线所示. 回路中 3 条边的容量的最小值 $\delta=2$. 记 h^C 为 C 上环流量 $\delta=2$ 的圈流，令 $f_1=f_0+h^C$，如图 7.10(a)所示. $N(f_1)$见图 7.10(b). 继续用 Floyd 算法检查 $N(f_1)$中是否有关于 aw 的负回路，计算如下. 由于 $d^{(t)}(i,i)=0(i=s,1,2,t)$，$N(f_1)$中没有关于 aw 的负回路，故 f_1是容量 $v_0=8$ 的最小费用流.

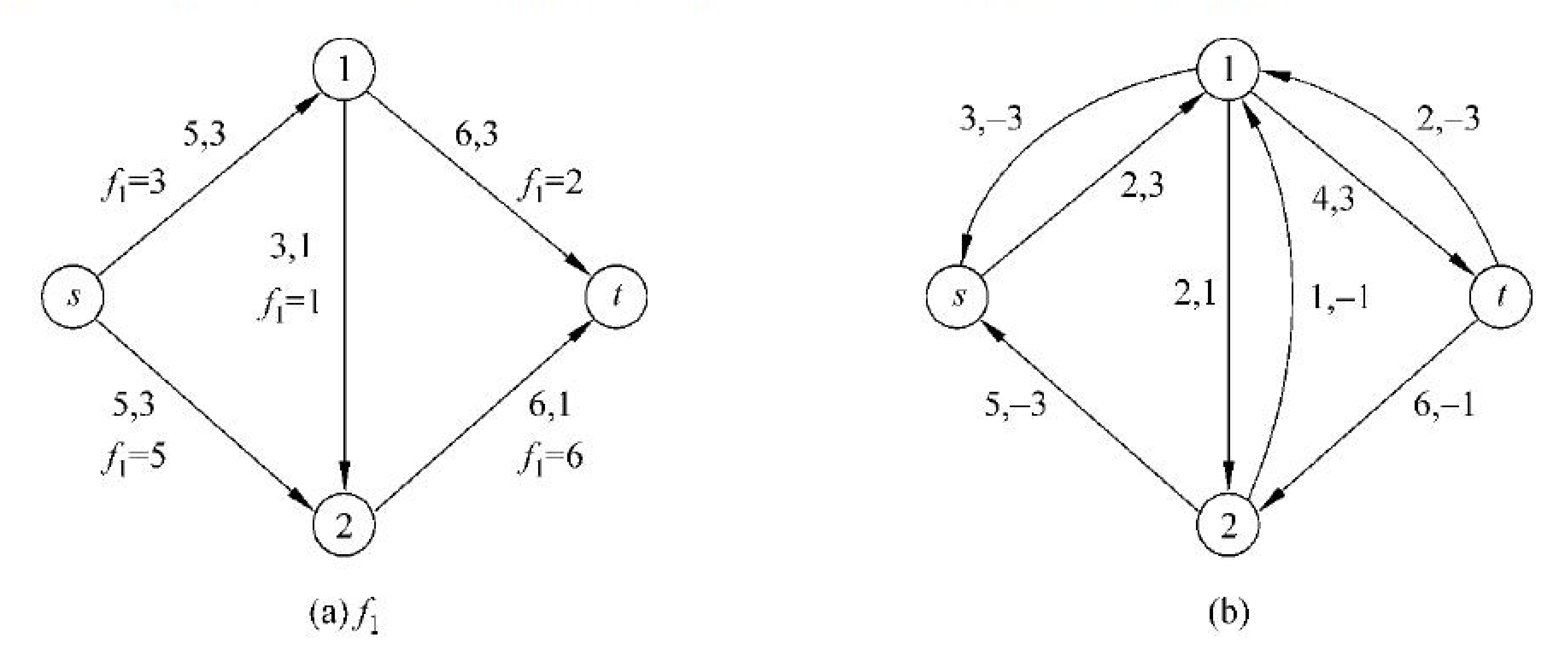


图 7.10　$N(f_1)$

$$d^{(0)}=\begin{bmatrix}0&3&+\infty&+\infty\\-3&0&1&3\\-3&-1&0&+\infty\\+\infty&-3&-1&0\end{bmatrix}\qquad h^{(0)}=\begin{bmatrix}s&1&0&0\\s&1&2&t\\s&1&2&0\\0&1&2&t\end{bmatrix}$$

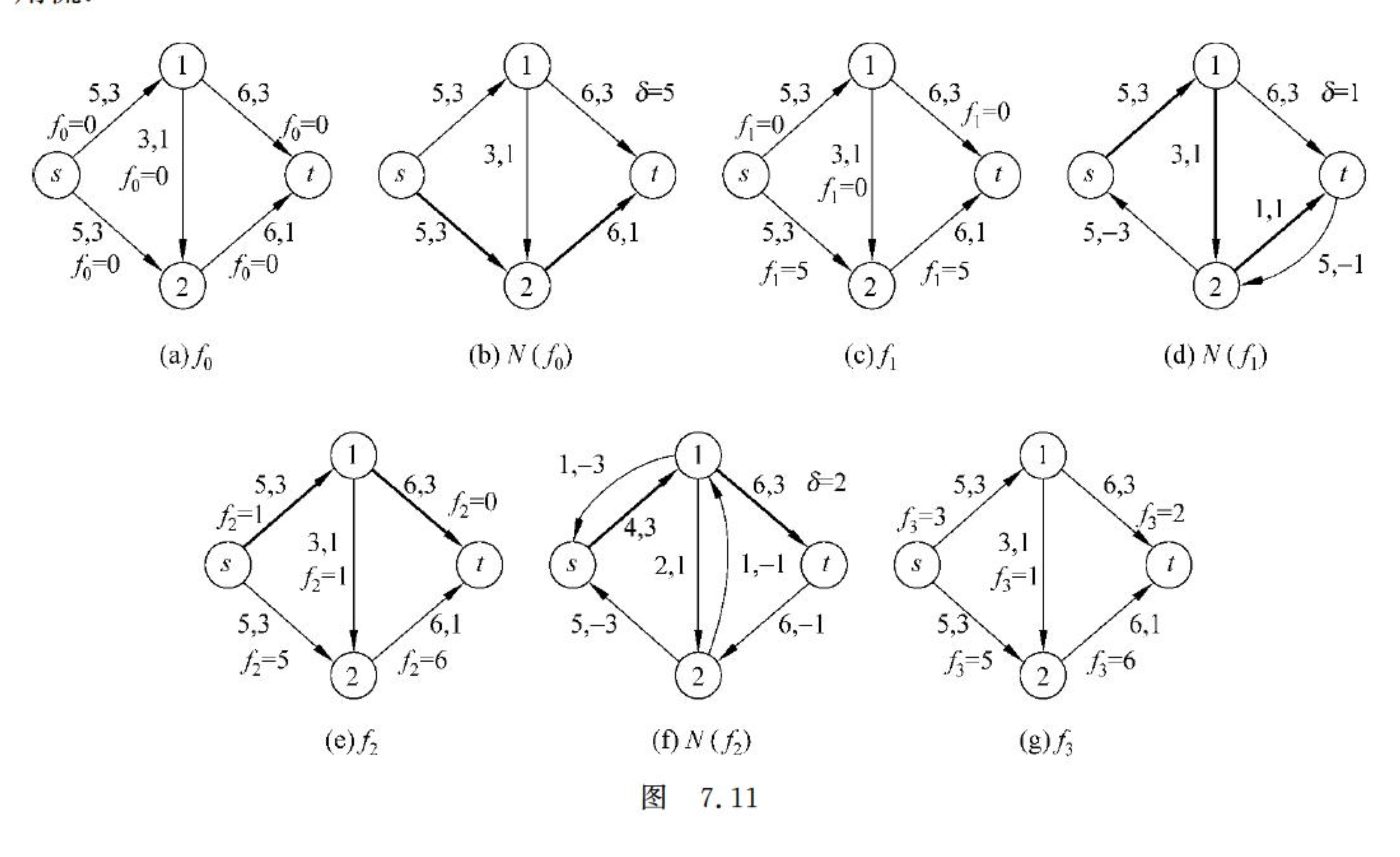
$$d^{(s)}=\begin{bmatrix}0&3&+\infty&+\infty\\-3&0&1&3\\-3&-1&0&+\infty\\+\infty&-3&-1&0\end{bmatrix}\qquad h^{(s)}=\begin{bmatrix}s&1&0&0\\s&1&2&t\\s&1&2&0\\0&1&2&t\end{bmatrix}$$

$$d^{(1)}=\begin{bmatrix}0&3&4&6\\-3&0&1&3\\-4&-1&0&2\\-6&-3&-2&0\end{bmatrix}\qquad h^{(1)}=\begin{bmatrix}s&1&1&1\\s&1&2&t\\1&1&2&1\\1&1&1&t\end{bmatrix}$$

$$d^{(2)}=\begin{bmatrix}0 & 3 & 4 & 6\\ -3 & 0 & 1 & 3\\ -4 & -1 & 0 & 2\\ -6 & -3 & -2 & 0\end{bmatrix}\qquad h^{(2)}=\begin{bmatrix}s & 1 & 1 & 1\\ s & 1 & 2 & t\\ 1 & 1 & 2 & 1\\ 1 & 1 & 1 & t\end{bmatrix}$$

$$d^{(t)}=\begin{bmatrix}0 & 3 & 4 & 6\\ -3 & 0 & 1 & 3\\ -4 & -1 & 0 & 2\\ -6 & -3 & -2 & 0\end{bmatrix}\qquad h^{(t)}=\begin{bmatrix}s & 1 & 1 & 1\\ s & 1 & 2 & t\\ 1 & 1 & 2 & 1\\ 1 & 1 & 1 & t\end{bmatrix}$$

7.15 用零流作为初始可行流 f_0,计算如图 7.11 所示. $N(f)$中关于权 aw 的 s-t 最短路径的计算过程略去,在图中用粗线给出. 在图 7.11(f)中 $N(f_2)$的最短路径上容量允许调整的流值是 4,但 $v_0-v(f_2)=2$,所以实际的调整量 $\delta=2$. f_3是所求的流量 $v_0=8$ 的最小费用流.



(a) f_0 (b) $N(f_0)$ (c) f_1 (d) $N(f_1)$ (e) f_2 (f) $N(f_2)$ (g) f_3

图 7.11

7.16 方法一:检查 FF 算法,标号过程都是从 s 开始到 t 结束,每个顶点最多标号一次,找到的每条 s-t 增广链都不会含有进入 s 的边和离开 t 的边,从而得到的最大流满足题中的要求.

方法二:设 f 是一个最大流,如果 $v(f)=0$,则零流满足题中的要求. 下面假设 $v(f)>0$. 在 $N=\langle V,E,c,s,t\rangle$上添加一个顶点 w 和两条边$\langle t,w\rangle$,$\langle w,s\rangle$,并且取 $c(t,w)=c(w,s)=v(f)$,得到容量网络 $N'=\langle V\cup\{w\},E\cup\{\langle t,w\rangle,\langle w,s\rangle\},c,s,t\}$. 令 $f'(t,w)=f'(w,s)=v(f)$,$\forall e\in E,f'(e)=f(e)$. f'是 N'上的可行流,$v(f')=0$ 且 f'不是零流. 根据引理 7.10,f'等于若干圈流的和. 设 $f'=\sum_{r=1}^{q}h^{C_r}$,其中每个 C_r是顶点不重复的回路. 记 f 是所有含 w 的回路 C_r上的圈流之和. 由于含 w 的回路 C_r不会含有 E 中进入 s 的边和离开 t 的边,故 f''在 E 中所有进入 s 的边和离开 t 的边上的值均为 0. 记 $f_{\max}$是 f''在

E 上的限制，$f_{\max}$是 N 上的可行流且在所有进入 s 的边和离开 t 的边上的值均为 0. 又由于 $f''(w,s)=v(f)$，有 $v(f_{\max})=v(f)$，$f_{\max}$也是 N 上的最大流，因此 $f_{\max}$满足题中的要求.

7.17 (1) 方法一：在算法 7.4 最小费用流的负回路算法中，把步骤 1 改为调用最大流算法，求得最大流 f.

方法二：在算法 7.5 最小费用流的最短路算法中，删去流量调整的上界 v.

算法如下：

```
1. f←0
2. 构造 N(f)
3. 调用 Ford 算法计算 N(f)中以 aw 为权的 s-t 最短路径
4. if N(f)中不存在 s-t 路径 then return f          //f 是最小费用最大流
5. 设求得的最短路径为 P,θ←min{ac(i,j)|<i,j>∈E(P)}
6. for <i,j>∈E(P) do
7.     if <i,j>∈E then f(i,j)← f(i,j)+θ
8.     else f(j,i)← f(j,i)−θ
9. goto 2
```

(2) 方法一：利用算法 7.4 最小费用流的负回路算法求解，对算法进行两点修改.

① 求一个最大流作为初始流.

② 设给定费用 w_0. 当 $w(f)\leqslant w_0$时，计算结束. 如果 $N(f)$中没有以 aw 为权的负回路且 $w(f)>w_0$，则不存在费用不超过 w_0的最大流.

方法二：如本题(1)中方法二，用最短路径算法求得最小费用最大流 f. 如果 $w(f)\leqslant w_0$，则 f 是费用不超过 w_0的最大流，输出 f；如果 $w(f)>w_0$，则不存在费用不超过 w_0的最大流.

7.18 这是运输问题，产销平衡. 用表上作业法，计算过程见表 7.5～表 7.9. 表 7.5 用最小元素法求初始方案 $x^{(0)}$，带圆圈的数字给出计算的顺序. 在表 7.7 中 $\lambda_{22}=-1$，对应的闭回路用虚线画出，$\delta=\min\{4,2.5\}=2.5$. 表 7.9 中 $x^{(1)}$ 的检验数都大于等于 0，故 $x^{(1)}$ 是最优调运方案.

表 7.5 初始方案 $x^{(0)}$

					a_i
	1 ④ [3]	4 ① [2]	[7]	[6]	~~5~~ ~~1~~
	2.5 ⑥ [7]	[5]	2 ② [2]	1.5 ⑤ [3]	~~6~~ ~~4~~ ~~2.5~~
	③ 2.5 [2]	[5]	[4]	[5]	~~2.5~~
b_j	~~6~~ ~~3.5~~ ~~2.5~~	~~4~~	~~2~~	~~1.5~~	

表 7.6 $x^{(0)}$ 的位势

	1 [3]	4 [2]	[7]	[6]	0
	2.5 [7]	[5]	2 [2]	1.5 [3]	4
	2.5 [2]	[5]	[4]	[5]	−1
v_j	3	2	−2	−1	

表 7.7 $x^{(0)}$ 的检验数

1 [0]	4 [0]	[9]	[7]
2.5 [0]	[−1]	2 [0]	1.5 [0]
2.5 [0]	[4]	[7]	[7]

表 7.8 $x^{(1)}$ 及其位势

					u_i
	3.5 [3]	1.5 [2]	[7]	[6]	0
	[7]	2.5 [5]	2 [2]	1.5 [3]	3
	2.5 [2]	[5]	[4]	[5]	−1
v_j	3	2	−1	0	

表 7.9　$x^{(1)}$ 的检验数

3.5 [0]	1.5 [0]	[8]	[6]
[1]	2.5 [0]	2 [0]	1.5 [0]
2.5 [0]	[4]	[6]	[6]

7.19　这个运输问题的产大于销，引入虚拟用户 B_5，其需求量等于产量与销量的差 4，使得产销平衡. 由于 B_5 是虚拟的，供给 B_5 的产品实际上是没有售出的产品，单位运费为 0，见表 7.10. 用最小元素法确定初始方案 $x^{(0)}$，见表 7.11. 在第④步，第 1 行和第 1 列的值都是 4，令 $x_{11}=4$，同时还要在第 1 行或第 1 列任取一个变量(它所在的行和列没有被划掉)令其为 0. 这里令 $x_{21}=0$，这个初始方案是退化的. 对 $x^{(0)}$ 的修改值 $\delta=0$. $x^{(0)}$ 和 $x^{(1)}$ 的值完全相同，只是把基变量 x_{21} 换成 x_{25}，把 x_{21} 位置上的 0 换到 x_{25} 处. $x^{(1)}\sim x^{(3)}$ 及其位势和检验数如表 7.12～表 7.19 所示。

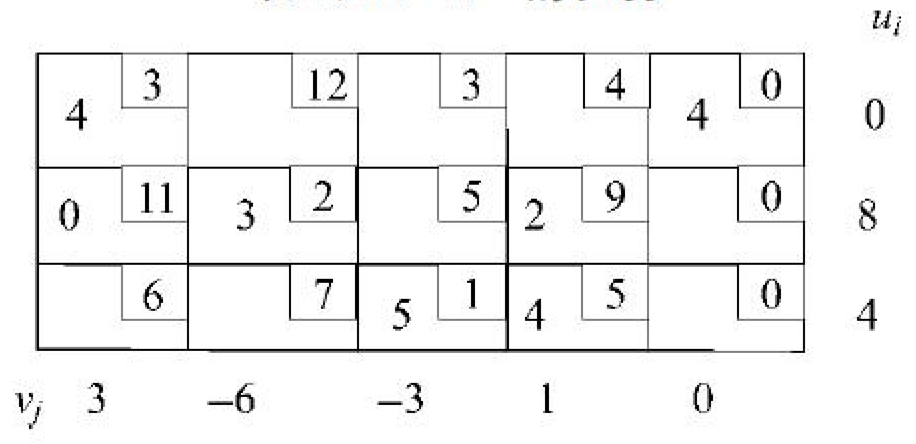
表 7.10　引入 B_5 使产销平衡

	用户 B_1	用户 B_2	用户 B_3	用户 B_4	用户 B_5	产量 a_i
基地 A_1	3	12	3	4	0	8
基地 A_2	11	2	5	9	0	5
基地 A_3	6	7	1	5	0	9
需求量 b_j	4	3	5	6	4	22

表 7.11　初始方案 $x^{(0)}$

						a_i
	4④ [3]	[12]	[3]	[4]	①4 [0]	~~8~~ 4
	0④ [11]	③3 [2]	[5]	2⑦ [9]	[0]	~~5~~ 2
	[6]	[7]	②5 [1]	4⑥ [5]	[0]	~~9~~ 4
b_j	~~4~~	~~3~~	~~5~~	~~6~~ 2	~~4~~	

表 7.12　$x^{(0)}$ 的位势



						u_i
	4 [3]	[12]	[3]	[4]	4 [0]	0
	0 [11]	3 [2]	[5]	2 [9]	[0]	8
	[6]	[7]	5 [1]	4 [5]	[0]	4
v_j	3	−6	−3	1	0	

表 7.13　$x^{(0)}$ 的检验数

4 [0]	[18]	[6]	[3]	4 [0]
0 [0]	3 [0]	[0]	2 [0]	[−8]
[−1]	[9]	5 [0]	4 [0]	[−4]

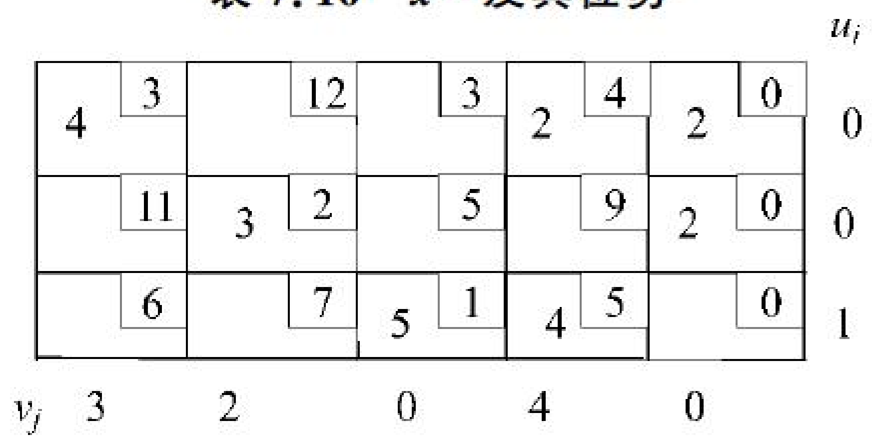
表 7.14　$x^{(1)}$ 及其位势

						u_i
	4 [3]	[12]	[3]	[4]	4 [0]	0
	[11]	3 [2]	[5]	2 [9]	0 [0]	0
	[6]	[7]	5 [1]	4 [5]	[0]	−4
v_j	3	2	5	9	0	

表 7.15　$x^{(1)}$ 的检验数

4 [0]	[10]	[−2]	[−5]	4 [0]
[8]	3 [0]	[0]	2 [0]	0 [0]
[7]	[9]	5 [0]	4 [0]	[4]

表 7.16　$x^{(2)}$ 及其位势



						u_i
	4 [3]	[12]	[3]	2 [4]	2 [0]	0
	[11]	3 [2]	[5]	[9]	2 [0]	0
	[6]	[7]	5 [1]	4 [5]	[0]	1
v_j	3	2	0	4	0	

表 7.17 $x^{(2)}$ 的检验数

4 / 0	10	3	2 / 0	2 / 0
8	3 / 0	5	5	2 / 0
2	4	5 / 0	4 / 0	–1

表 7.18 $x^{(3)}$ 及其位势

					u_i
4 / 3	12	3	4 / 4	0	0
11	3 / 2	5	9	2 / 0	1
6	7	5 / 1	2 / 5	2 / 0	1
v_j: 3	1	0	4	–1	

考虑到 B_5 是虚拟用户，构造初始方案时最后考虑 B_5 可能更好些. 仍采用最小元素法，这样得到的初始方案就是上面的 $x^{(3)}$，直接得到最优解，如表 7.20 所示。

表 7.19 $x^{(3)}$ 的检验数

4 / 0	11	3	4 / 0	1
7	3 / 0	4	4	2 / 0
2	5	5 / 0	2 / 0	2 / 0

表 7.20 初始方案 $x^{(0)}$

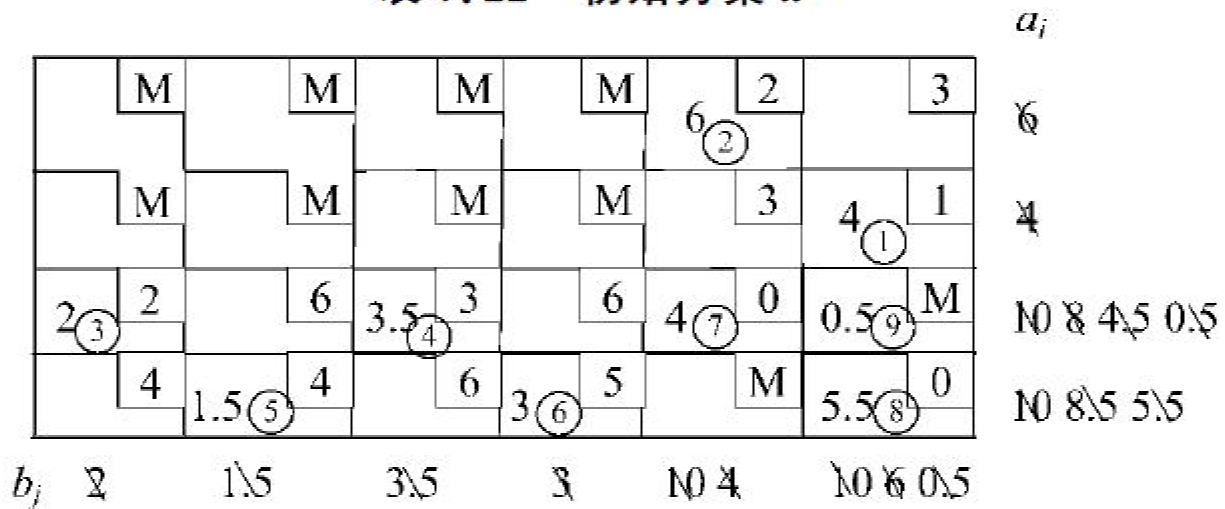
					a_i
4③ / 3	12	3	4④ / 4	0	8 4
11	3② / 2	5	9	2⑥ / 0	5 2
6	7	5① / 1	2⑤ / 5	2⑦ / 0	9 4 2
b_j: 4	3	5	6 2	4 2	

7.20 仓库既要运入又要运出，所以把它们既看作产地又看作销地. 于是，有 4 个产地 A_1，A_2，C_1 和 C_2，6 个销地 B_1，B_2，B_3，B_4，C_1 和 C_2. 总产量等于总销量等于 10，仓库的进出量不可能事先知道，但不会超过 10，可把仓库的产量和销量都定为 10. C_1 到 C_1 的单位运费为 0，因为实际上不存在这部分运输. 而 C_1 不允许运送到 C_2，故单位运费为 M(充分大). 同样地，C_2 到 C_2 的单位运费为 0，C_2 到 C_1 的单位运费为 M. 同理，$A_i(i=1,2)$ 到 $B_j(j=1,2,3,4)$ 的单位运费为 M. 如表 7.21 所示. 用最小元素法构造初始方案，考虑到 C_1 到 C_1 和 C_2 到 C_2 是不存在的，把这两个值放在后面确定. $x^{(0)}$ 的检验数有 4 个是负的，取 $x_{34}(\lambda_{34}=-M+1)$ 作为换入变量，换出变量是 x_{36}，调整量 $\delta=0.5$. $x^{(1)}$ 的检验数都是非负的，$x^{(1)}$ 是最优运输方案，如表 7.22～表 7.26 所示.

表 7.21 带中转站的运输模型

	B_1	B_2	B_3	B_4	C_1	C_2	产量 a_i
A_1	M	M	M	M	2	3	6
A_2	M	M	M	M	3	1	4
C_1	2	6	3	6	0	M	10
C_2	4	4	6	5	M	0	10
销量 b_j	2	1.5	3.5	3	10	10	30

表 7.22 初始方案 $x^{(0)}$



						a_i
M	M	M	M	6② / 2	3	6
M	M	M	M	3	4① / 1	4
2③ / 2	6	3.5④ / 3	6	4⑦ / 0	0.5⑨ / M	10 8 4.5 0.5
4	1.5⑤ / 4	6	3⑥ / 5	M	5.5⑧ / 0	10 8.5 5.5
b_j: 2	1.5	3.5	3	10 4	10 6 0.5	

表 7.23　$x^{(0)}$ 及其位势

							u_i
	[M]	[M]	[M]	[M]	6 [2]	[3]	0
	[M]	[M]	[M]	[M]	[3]	4 [1]	-M-1
	2 [2]	[6]	3.5 [3]	[6]	4 [0]	0.5 [M]	-2
	[4]	1.5 [4]	[6]	3 [5]	[M]	5.5 [0]	-M-2
v_j	4	M+6	5	M+7	2	M+2	

表 7.24　$x^{(0)}$ 的检验数

[M-4]	[-6]	[M-5]	[-7]	6 [0]	[-M+1]
[2M-3]	[M-5]	[2M-4]	[M-6]	[M+2]	4 [0]
2 [0]	[-M+2]	3.5 [0]	[-M−1]	4 [0]	0.5 [0]
[M+2]	1.5 [0]	[M+3]	3 [0]	[2M]	5.5 [0]

表 7.25　$x^{(1)}$ 及其位势

							u_i
	[M]	[M]	[M]	[M]	6 [2]	[3]	0
	[M]	[M]	[M]	[M]	[3]	4 [1]	−2
	2 [2]	[6]	3.5 [3]	0.5 [6]	4 [0]	[M]	−2
	[4]	1.5 [4]	[6]	2.5 [5]	[M]	6 [0]	−3
v_j	4	7	5	8	2	3	

表 7.26　$x^{(1)}$ 的检验数

[M-4]	[M-7]	[M-5]	[M-8]	6 [0]	[0]
[M-2]	[M-5]	[M-3]	[M-6]	[3]	4 [0]
2 [0]	[1]	3.5 [0]	0.5 [0]	4 [0]	[M-1]
[3]	1.5 [0]	[4]	2.5 [0]	[M+1]	6 [0]

7.21　(1) 每个季度是一个发点、同时又是一个收点，这是一个多发点、多收点的最小费用流问题. 引入一个超级发点和一个超级收点，问题转化为最小费用最大流问题，容量-费用网络如图 7.12 所示，边旁的第 1 个数是容量，第 2 个数是费用.

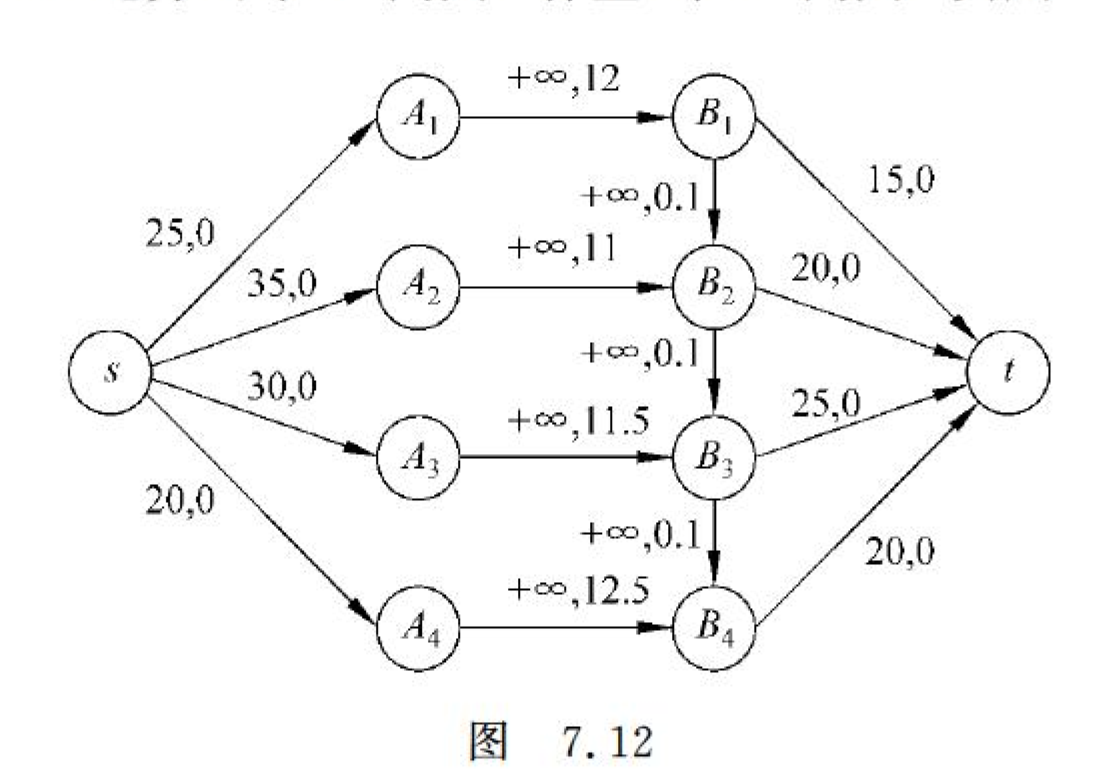


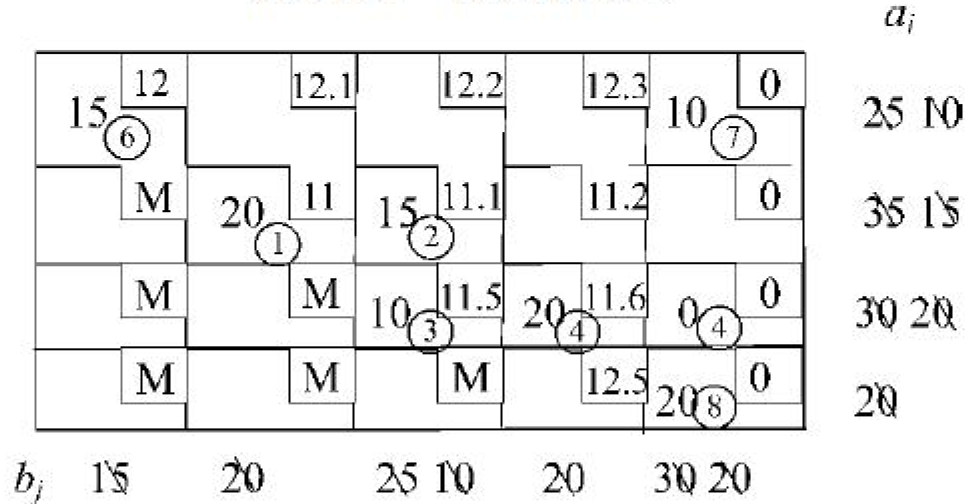
图　7.12

(2) 表 7.27 给出问题的运输模型. 每个季度是一个产地，同时又是一个销地，生产能力是产量，交货量是销量. 总生产能力 110 大于总交货量 80，引入一个虚拟销地(剩余能力)，其销量为 30. 用最小元素法确定初始方案，但把虚拟销地放在后面. 在第 4 步确定 x_{34} 时，行列的剩余值都是 20，故除取 $x_{34}=20$ 外，还要在第 3 行或第 4 列填一个 0，这里取 $x_{35}=0$. $x^{(0)}$ 的检验数都非负，故 $x^{(0)}$ 是使成本最小的生产计划. 如表 7.28～表 7.30 所示.

表 7.27　带中转站的运输模型

	1 季度	2 季度	3 季度	4 季度	剩余能力	生产能力 a_i
1 季度	12.0	12.1	12.2	12.3	0	25
2 季度	M	11.0	11.1	11.2	0	35
3 季度	M	M	11.5	11.6	0	30
4 季度	M	M	M	12.5	0	20
交货量 b_j	15	20	25	20	30	110

表 7.28　初始方案 $x^{(0)}$



						a_i
	15⑥ [12]	[12.1]	[12.2]	[12.3]	10⑦ [0]	25 10
	[M]	20① [11]	15② [11.1]	[11.2]	[0]	35 15
	[M]	[M]	10③ [11.5]	20④ [11.6]	0④ [0]	30 20
	[M]	[M]	[M]	[12.5]	20⑧ [0]	20
b_j	15	20	25 10	20	30 20	

表 7.29　$x^{(0)}$ 的位势

						u_i
	15 [12]	[12.1]	[12.2]	[12.3]	10 [0]	0
	[M]	20 [11]	15 [11.1]	[11.2]	[0]	−0.4
	[M]	[M]	10 [11.5]	20 [11.6]	0 [0]	0
	[M]	[M]	[M]	[12.5]	20 [0]	0
v_j	12	11.4	11.5	11.6	0	

表 7.30　$x^{(0)}$ 的检验数

15 [0]	[0.7]	[0.7]	[0.7]	10 [0]
[M-11.6]	20 [0]	15 [0]	[0]	[0.4]
[M-12]	[M-11.4]	10 [0]	20 [0]	0 [0]
[M-12]	[M-11.4]	[M-11.5]	[0.9]	20 [0]

7.22　作二部图 $G=\langle X,Y,E\rangle$，其中 $X=\{\mathrm{A,B,C,D,E}\}$，$Y=\{1,2,3,4,5\}$，$E=\{(x,y)\mid x$ 申请岗位 $y\}$，如图 7.13(a)所示. 问题是求 G 的最大匹配. 用匈牙利算法求解. 前 3 个阶段很容易地得到匹配 $M_1=\{(\mathrm{A},1),(\mathrm{B},4),(\mathrm{C},5)\}$，见图 7.13(b)中的粗线. 接下来经过标号找到增广交错路径 $P=\mathrm{D4B1A2}$，$M_2=M_1\oplus P=\{(\mathrm{A},2),(\mathrm{B},1),(\mathrm{C},5),(\mathrm{D},4)\}$ 见图 7.13(c)中的粗线. 图 7.13(c)中的标号表明 M_2 是 G 的最大匹配. 分配方案如下：A 分配岗位 2，B 分配岗位 1，C 分配岗位 5，D 分配岗位 4. E 这次没有岗位.

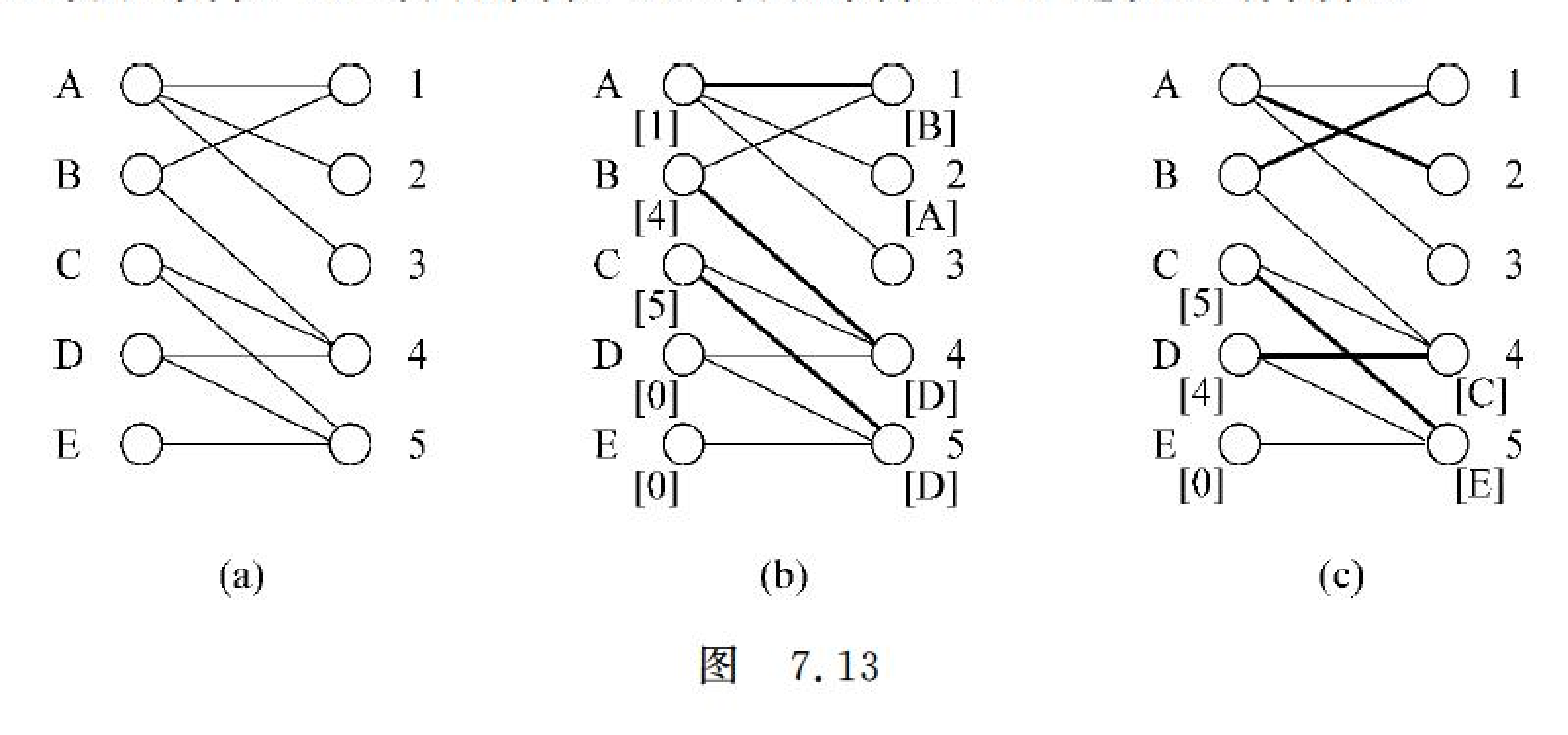


图　7.13

7.23　用匈牙利算法. 前 3 阶段得到 $M_1=\{(A_2,B_1),(A_3,B_3),(A_5,B_2)\}$，第 4 阶段如

图7.14(a)所示. 带单圈的表示 $w_{ij}-\alpha_i-\beta_j=0$,对应一条边. 双圈对应一条匹配边. 图中顶点旁方括号内是标号,打勾的行、列对应已标号顶点,未打勾的行、列对应未标号顶点. $s_j=\min\{w_{ij}-\alpha_i-\beta_j \mid A_i\text{行打勾},B_j\text{列未打勾}\}$,$h_j$是第 j 列取值 s_j 的行 A_i. θ 等于诸 s_j 的最小值的一半. 对 α,打勾的行加 θ,不打勾的行减 θ; 对 β,打勾的列减 θ,不打勾的列加 θ. 调整图7.14(a)中的 α 和 β 后,得到图7.14(b). 设 s 在 B_j 列取到最小值,对应的 h 值是 A_i,则对于调整后的 α 和 β,$w_{ij}-\alpha_i-\beta_j=0$. 在图中添加新的边$(A_i,B_j)$,这里是$(A_1,B_5)$. 见图7.14(b). 经过标号得到新的匹配边$(A_1,B_5)$,$M_2=\{(A_1,B_5),(A_2,B_1),(A_3,B_3),(A_5,B_2)\}$,见图7.14(c). 调整图7.14(c)中的 α 和 β,得到图7.14(d). 添加新边(A_4,B_1),标号及 s,h 的计算结果见图7.14(d). 再调整 α、β,添加新边(A_2,B_4)和(A_4,B_5),经过标号找到增广交错路径 $P=A_4B_1A_2B_4$. 最小权完美匹配 $M_3=M_2\oplus P=\{(A_1,B_5),(A_2,B_4),(A_3,B_3),(A_4,B_1),(A_5,B_2)\}$,见图7.14(f).

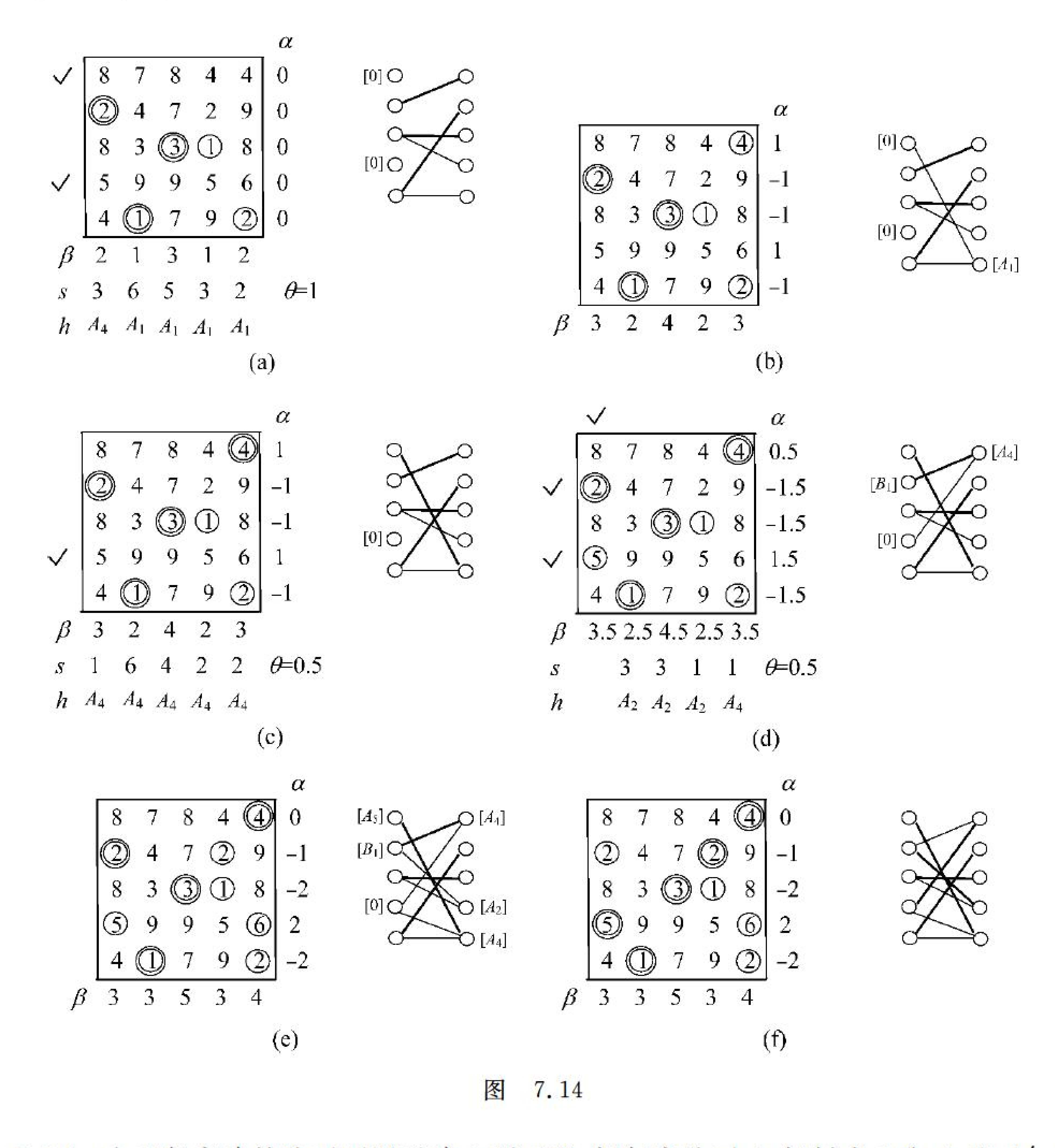


图 7.14

7.24 由于每家建筑公司可以承建2项工程,把每家公司 A_i 复制成2个 A_i 和 A_i',每个承建一项工程. 这样共有6个公司、5项工程,添加一项虚拟工程(造价均为0),构成一个指

派问题,完全二部图的权函数如图 7.15 所示.

10	15	9	8	10	0
10	15	9	8	10	0
9	18	6	10	8	0
9	18	6	10	8	0
6	14	8	8	6	0
6	14	8	8	6	0

图 7.15

最小权完美匹配是 $M=\{(A_1,B_4),(A'_1,B_2),(A_2,B_6),(A'_2,B_3),(A_3,B_1),(A'_3,B_5)\}$,即 A_1 承建工程 B_2 和 B_4,A_2 承建工程 B_3(B_6 是虚拟工程),A_3 承建工程 B_1 和 B_5,工程总造价为 15+8+6+6+6=41.

7.25 直接用公式(7.20)计算 θ 需要时间 $O(n^2)$. 在一个阶段中修改 α_i 和 β_j(即执行 5~5.8)的次数是 $O(n)$. 这样一来,每个阶段的时间为 $O(n^3)$,算法的时间复杂度为 $O(n^4)$.

7.26 显然,$\sum_{x\in A}b(x)=\sum_{y\in B}b(y)$ 是存在 b-匹配的必要条件.

二部图 b-匹配算法如下:

1. 若 $v_0=\sum_{x\in A}b(x)\neq\sum_{y\in B}b(y)$,则输出"不存在 b-匹配",计算终止.
2. 构造容量网络 $N=<V,E',c,s,t>$,其中:

$$V=A\cup B\cup\{s,t\}$$

$$E'=\{<x,y>\mid(x,y)\in E\}\cup\{<s,x>\mid x\in A\}\cup\{<y,t>\mid y\in B\}$$

$$c(u,v)=\begin{cases}b(v), & u=s\wedge v\in A\\ b(u), & u\in B\wedge v=t\\ 1, & (u,v)\in E\end{cases}$$

3. 用 Dinic 算法求 N 的最大流,设 f 是求得的最大流.
4. 若 $v(f)=v_0$,则输出 $M=\{(x,y)\mid f(x,y)=1,(x,y)\in E\}$;否则,输出"不存在 b-匹配".

不难证明,当 $\sum_{x\in A}b(x)=\sum_{y\in B}b(y)$ 时,M 是 G 的 b-匹配当且仅当 N 有最大流 f,其中

$$f(u,v)=\begin{cases}b(v), & u=s\wedge v\in A\\ b(u), & u\in B\wedge v=t\\ 1, & (u,v)\in M\\ 0, & \text{否则}\end{cases}$$

因此,算法是正确的.

记 $n=|A|+|B|$,步骤 1 需要时间 $O(n)$,步骤 2 和步骤 4 的时间为 $O(n^2)$,而步骤 3 需要 $O(n^3)$,故算法的时间复杂度为 $O(n^3)$.

7.27 容易证明下述命题,

命题 完美匹配 M 是最大权最小的完美匹配当且仅当 $G(w^*)$ 没有完美匹配,其中 $G(w^*)=<A,B,E(w^*)>$,$w^*=\max\{w(e)\mid e\in M\}$,$E(w^*)=\{e\mid w(e)<w^*,e\in E\}$.

根据这个命题,下述算法是正确的.

二部图瓶颈匹配算法如下:

1. 求 G 的一个最大匹配 M.
2. 若 M 不是完美匹配,则输出"G 没有完美匹配",计算结束.
3. $w^*\leftarrow\max\{w(e)\mid e\in M\}$.
4. $E'\leftarrow E-\{e\mid w(e)\geq w^*\}$,$M'\leftarrow M-\{e\mid w(e)\geq w^*\}$

5. 以 M' 为初始匹配,用匈牙利算法求 $G'=<A,B,E'>$ 的最大匹配 M.

6. 若 M 是 G 的完美匹配,则返回步骤 3. 否则 M 是最大权最小的完美匹配,输出 M.

记 G 的顶点数 n,边数 m. 步骤 1 和步骤 5 的时间为 $O(nm)$. 步骤 3～步骤 6,每次至少删去一条边,重复的次数一定小于 m. 算法的时间复杂度为 $O(nm^2)$.

7.28 (1) 因为 M 中的边是不相交的,覆盖 M 中的边需要 $|M|$ 个顶点,故 $|M|\leqslant|V|$.

(2) 每一条边 (x,y),$x\in A$,$y\in B$,有 4 种可能:

① x 未标号,y 已标号.

② x 已标号,y 已标号.

③ x 未标号,y 未标号.

④ x 已标号,y 未标号.

前 3 种边都被 V 中的顶点覆盖,只需证明第④种不可能发生. 设 x 已标号,y 未标号,根据标号的规则,$(x,y)\in M$,否则因为 x 已标号,y 可从 x 得到标号. 于是,x 是饱和点,它的标号只能从 B 中某个已标号的顶点得到,即存在已标号的 $y'\in B$,使得 $(x,y')\in M$. 而与 x 关联的匹配边只有一条,故 $y=y'$. y 未标号,y' 已标号,矛盾. 得证 V 是一个顶点覆盖.

$V=A_1\cup B_1$. 因为 M 是最大匹配,B_1 中的顶点都是饱和点. A 的非饱和点都是已标号的,故 A_1 中的顶点也是饱和点. 又任意的 $x\in A_1$ 和 $y\in B_1$,$(x,y)M$,否则 x 可以从 y 得到标号. 因此,V 中任意两个顶点关联两条不同的匹配边,故 $|V|\leqslant|M|$. 由(1),得证 $|V|=|M|$. 从而,V 是最小顶点覆盖.

(3) 根据本题上述(2),在用匈牙利算法求得二部图最大匹配的同时,也得到了最小顶点覆盖.

第 8 章 算法分析与问题的计算复杂度

8.1 内容提要

1. 基本概念

问题的时间复杂性上界 在此时间内，一定能求出该问题的解. 该问题的任何一个算法在最坏情形下的时间复杂性都给出了该问题的时间复杂性的一个上界.

问题的时间复杂性下界 求解该问题需要的最少的时间，即如少于该时间，一定存在某个实例不能求出该实例的解.

问题的时间复杂性紧下界 对于该问题的一个时间复杂性紧下界，如果存在一个该问题的算法，其时间复杂性函数恰好为该下界，则称此下界为该问题的紧下界.

问题的时间复杂度 求解该问题需要的最少的时间，并且在该时间内一定能解该问题. 这是问题的一个固有性质.

2. 某些重要结果

以比较作为基本运算，检索问题的最坏情形时间复杂度为 $\log n+1$.

以数的比较作为基本运算，排序问题最坏情形时间复杂度为 $\Theta(n\log n)$. 排序问题的各种主要算法的优劣如表 8.1 所示.

表 8.1 有关排序算法的分析结果

算 法	最坏情况	平均情况	占用空间	时间上的最优性
冒泡排序	$O(n^2)$	$O(n^2)$	原地	
快速排序	$O(n^2)$	$O(n\log n)$	$O(\log n)$	平均时间最优
归并排序	$O(n\log n)$	$O(n\log n)$	$O(n)$	最优
堆排序	$O(n\log n)$	$O(n\log n)$	原地	最优

以数的比较作为基本运算，选择问题最坏情况的时间复杂度为 $\Theta(n)$. 选择问题的各种主要算法的优劣如表 8.2 所示.

表 8.2 选择算法的时间复杂度

问 题	算 法	最坏情况	时间复杂性	最 优 性
找最大	Findmax	$n-1$	$n-1$	最优
找最大最小	FindMaxMin	$3n/2-2$	$3n/2-2$	最优

续表

问　题	算　法	最坏情况	时间复杂性	最　优　性
找第二大	锦标赛	$n+\log n-2$	$n+\log n-2$	最优
找中位数	Select	$O(n)$	$3n/2-3/2$	阶最优
找第 k 小	Select	$O(n)$	$n+\min\{k,n-k+1\}-2$	阶最优

8.2　习　题

8.1　设 $G_1=<V_1,E_1>$,$G_2=<V_2,E_2>$是两个简单图,其中 $V_1=V_2$. 假设 G_1 和 G_2 的输入是用邻接矩阵表示的. 换句话说,如果(v_i,v_j)是图的边,那么它的邻接矩阵 M 的第 i 行第 j 列的元素 $r_{ij}=1$;否则 $r_{ij}=0$,$1\leqslant i<j\leqslant n$.

(1) 设输入规模是图中的顶点数 n,给出一个算法判定 G_1 是否为 G_2 的补图. 说明算法的设计思想,并给出最坏情况下的时间复杂度.

(2) 对于求解这个问题的所有算法,给出一个尽可能紧的时间复杂度的下界,并证明你的结果.

8.2　对于给定的 $x\neq 0$,求 n 次多项式 $P(x)=a_0+a_1x+a_2x^2+\cdots+a_nx^n$ 的值.

(1) 设计一个在最坏情况下时间复杂度为 $\Theta(n)$的求值算法.

(2) 证明任何求值算法的时间复杂度都是 $\Omega(n)$.

8.3　(1) 设 A 和 B 是两个长为 n 的有序数组,现在需要将 A 与 B 合并成一个排好序的数组,证明任何以元素比较作为基本运算的归并算法至少要做 $2n-1$ 次比较.

(2) 对上述归并问题,假设$|A|=m$,$|B|=n$,给出求解该问题的最优算法并证明其最优性.

8.4　设 n 是 k 的倍数,有 k 个排好序的数表 $L_1,L_2,\cdots,L_k$,每个数表都有 n/k 个数. 假设 n 个数彼此不等,并且归并长为 m,n 的两个数表的时间代价是 $O(m+n)$.

(1) 使用顺序归并算法归并这 k 个数表,在最坏情况下的时间复杂度是什么?

(2) 设计一个时间复杂度更低的归并算法,说明算法的主要设计思想,并分析你的算法在最坏情况下的时间复杂度.

(3) 对于以比较作为基本运算求解上述问题的算法类,最坏情况下的时间复杂度的下界是什么? 证明你的结果.

8.5　求直线点对问题的一个紧的下界.

8.6　以 $n=4$ 为例画出冒泡排序算法的决策树.

8.7　设 A 是 n 个不等的整数按照递增次序排列的数组,已知存在 $i\in\{1,2,\cdots,n\}$使得 $A[i]=i$,问怎样找到 i?

(1) 设计一个算法求解上述问题,给出算法的伪码描述并分析算法在最坏情况下的时间复杂度.

(2) 证明任何求解上述问题的算法至少需要做 $\Omega(\log n)$次比较.

8.8　设 S 是 n 个数构成的数组,判断 S 中的元素是否都是唯一的. 如果唯一,则输出"Yes";否则输出"No". 证明唯一性判定问题的复杂度是 $\Theta(n\log n)$.

8.9　设 $L=\{a_1,a_2,\cdots,a_n\}$是 n 个不相等的实数的数表,m 是小于 n 的正整数. 现在需

要按照从小到大的次序输出 L 中最小的 m 个数.

(1) 如果 $m=\Theta(n/\log n)$,以 L 中元素的比较作为基本运算,设计一个 $O(n)$时间的算法.

(2) 如果 $m=\omega(n/\log n)$,证明不存在 $O(n)$时间的算法.

8.10 证明任何从 n 个数中选第 k 小的数的算法,如果以比较作为基本运算,那么它至少要做 $n+\min\{k,n-k+1\}-2$ 次比较.

8.11 给定平面上 n 个点的坐标. 在这些点之间存在某些边,边 (i,j) 的权值是点 i 和 j 的距离. 这些点和边构成平面上的简单图 G,求 G 的一棵最小生成树. 证明求解该问题的算法类的时间复杂度下界是 $\Omega(n\log n)$.

8.3 习题解答与分析

8.1 (1) 设 G_1 与 G_2 的矩阵是 $\boldsymbol{M}_1=(a_{ij})$ 和 $\boldsymbol{M}_2=(b_{ij})$.

算法

1. 对于每个 $1\leqslant i<j\leqslant n$,比较 a_{ij} 和 b_{ij}.

2. 如果存在 $a_{ij}=b_{ij}$,那么 G_1 不是 G_2 的补图;如果都不等,则 G_1 是 G_2 的补图.

$$W(n)=n(n-1)/2$$

(2) **证明一.** $\boldsymbol{M}_1$ 和 $\boldsymbol{M}_2$ 的 i 行 j 列的元素分别记为 a_{ij} 和 b_{ij}. 任何求解该问题的判定算法都需要检查 $\boldsymbol{M}_1$ 与 $\boldsymbol{M}_2$ 上三角区域的所有 a_{ij} 与 b_{ij}. 如果有一项 a_{ij} 和 b_{ij} 没有做检查,那么对于 $a_{ij}=b_{ij}$ 而其他值都不等的两个输入矩阵 $\boldsymbol{M}_1$ 和 $\boldsymbol{M}_2$,算法不能加以区分. 而这两种情况的输出恰好相反,算法就会出错. 矩阵的上三角区域有 $n(n-1)/2$ 项,因此算法的时间复杂度至少是 $n(n-1)/2$.

证明二. 考虑决策树,构造方法如下.

对于任意算法 A:

1. 若 A 第一步考察 a_{ij} 和 b_{ij},将根结点标记为 (i,j).

2. 假设 A 标记了树中的结点 (i,j). 当 $a_{ij}=b_{ij}$ 时,算法进入左子树停机,并把左儿子记作树叶,标记为对应的输入. 当 $a_{ij}\neq b_{ij}$ 时,如果 A 下一步考察的是 a_{kl} 和 b_{kl},那么将 (i,j) 结点的右儿子标记为 (k,l).

3. 重复第 2 步,直到所有的输入都被标记为止.

对于不同的输入,算法在叶结点停止. 在最坏情况下,算法执行的操作数等于树的深度. 问题的输入是两个图的邻接矩阵,算法可区分的不同的输入有 $2^{n(n-1)/2}$ 种,每个输入对应于决策树的一片树叶,而具有 $2^{n(n-1)/2}$ 片树叶的二叉树的深度至少是 $n(n-1)/2$.

8.2 (1) 迭代计算:

$$\begin{aligned}
P_1(x) &= a_n \\
P_2(x) &= a_{n-1}+P_1(x)\cdot x \\
P_3(x) &= a_{n-2}+P_2(x)\cdot x \\
&\vdots \\
P_{n+1}(x) &= a_0+P_n(x)\cdot x
\end{aligned}$$

不难看出

$$P_{n+1}(x)=a_0+a_1x+a_2x^2+\cdots+a_nx^n=P(x)$$

如果顺序计算多项式 $P_1(x),P_2(x),\cdots,P_{n+1}(x)$，最后得到的就是 $P(x)$. 在这个过程中，计算 $P_i(x)$需要用到 $P_{i-1}(x)$的值，而且只需要 1 次乘法和 1 次加法，是常数时间. 于是计算整个多项式序列仅需要 $\Theta(n)$时间.

(2) **证** 多项式 $P(x)$有 $n+1$ 个系数，每个系数至少需要被处理 1 次. 如若不然，假设某个求值算法 A 不对系数 a_i 进行处理. 考虑输入

$$\langle a_0,a_1,\cdots,a_{i-1},a_i,a_{i+1},\cdots,a_n\rangle \text{ 和 } \langle a_0,a_1,\cdots,a_{i-1},b_i,a_{i+1},\cdots,a_n\rangle$$

其中 $a_i \neq b_i$ 是不为 0 的数，其他系数都相等. 那么这两个多项式对同一个 x 的求值一定不等，但是算法 A 的输出是不加区别的. 因此算法 A 出错. 于是任何算法至少需要 $n+1$ 次运算，即在最坏情况下时间复杂度为 $\Omega(n)$.

8.3 (1) **证** 设输入 $A=\{a_1,a_2,\cdots,a_n\}$，$B=\{b_1,b_2,\cdots,b_n\}$，满足

$$a_1<b_1<a_2<b_2<\cdots<a_i<b_i<a_{i+1}<\cdots<a_n<b_n$$

设 T 是任意归并算法. 对于任何 $a_i\in A$，T 都需要把 a_i 和 b_{i-1} 及 b_i 进行比较. 如果对于某个 i，算法 T 没有执行 a_i 和 b_{i-1} 的比较，将对满足上述条件的输入 I_1 和 I_2 没办法区分，其中

$$I_1:A_1=\{a_1,a_2,\cdots,a_{i-1},a_i,a_{i+1},\cdots,a_n\},\quad B_1=\{b_1,b_2,\cdots,b_{i-2},b_{i-1},b_i,\cdots,b_n\}$$

$$I_2:A_2=\{a_1,a_2,\cdots,a_{i-1},b_{i-1},a_{i+1},\cdots,a_n\},\quad B_2=\{b_1,b_2,\cdots,b_{i-2},a_i,b_i,\cdots,b_n\}$$

同理，a_i 与 b_i 的比较也是必不可少的. 由于 A 中含有 n 个数，除了 a_1 只需要与 b_1 比较 1 次之外，其他 $n-1$ 个 a_i 都要做 2 次比较，因此至少需要比较 $2n-1$ 次.

(2) 不妨设 $m>n$. 当 $n=\Theta(m)$时可采用顺序归并的算法，伪码如下：

```
算法 Merge(A,B)
输入：有序数组 A[1..m],B[1..n]
输出：C[1..m+n]
p←1,q←1,i←1.
while p≤m and q≤n do
    if A[p]<B[q]
    then C[i]←A[p],p←p+1,i←i+1
    else C[i]←B[q],q←q+1,i←i+1
if p>m
then 将 B 的剩余元素顺序放到 C[i..m+n]
else 将 A 的剩余元素顺序放到 C[i..m+n]
```

以比较作为基本运算，算法 Merge 在最坏情况下的比较次数是 $m+n-1$.

当 $n=O(\log m)$时，可以采用另一种算法：对 B 中的每个元素通过二分检索找到其在 A 中的合适位置，然后将它插入. 以比较做基本运算，该算法在最坏情况下的时间复杂度是 $O(\log^2 m)$.

下面证明算法的最优性. 不妨设 A 与 B 中的数都不相等. 合并后的数组含 $m+n$ 个元素. 每一个输入相当于从 $m+n$ 个位置中选 m 个位置来放 A 中的元素，不同的输入个数是 $C(m+n,m)$. 构造决策树，结点(i,j)表示将 a_i 和 b_j 进行比较. 如果 $a_i<b_j$，则将算法下一次比较的标记作为(i,j)的左儿子；否则作为右儿子. 算法结束时的树叶标记为输入，树叶片数是输入的个数，即 $C(m+n,m)$. 决策树中的树深代表最坏情况下的比较次数. 由于具有 $C(m+n,m)$片树叶的二叉树的深度至少是 $\log(C(m+n,m))$，即

$$\log\binom{m+n}{m}=\log\frac{(m+n)!}{m!n!}=\Theta((m+n)\log(m+n)-m\log m-n\log n)$$

当 $m=cn$ 时，上述下界变成

$$\Theta\left(m\log\left(1+\frac{1}{c}\right)+n\log(1+c)\right)$$

与算法时间复杂度的阶是一致的. 若 $n=O(\log m)$，则上述下界是

$$(m+n)\log(m+n)-m\log m-n\log n=m\log\frac{m+n}{m}+n\log\frac{m+n}{n}>n\log\frac{m+n}{n}$$

$$>n\log\frac{m}{n}=\Theta(n\log m)=\Theta(\log^2 m)$$

与算法时间复杂度的阶是一致的.

8.4 （1）顺序归并算法. 伪码如下：

1. $L\leftarrow L_1$
2. for $j\leftarrow 2$ to k
3. 　$L\leftarrow L\cup L_j$　　//L 与 L_j 归并得到新的 L
4. return L

在归并过程中 L_1 中的元素被比较 $k-1$ 次，L_2 的元素被比较 $k-1$ 次，…，L_{k-1} 的元素被比较 2 次，L_k 的元素被比较 1 次. 总共比较

$$n/k[(k-1)+(k-1)+(k-2)+\cdots+2+1]=n(k^2+k-2)/2k=O(kn)$$

（2）使用二分归并算法. 伪码如下：

1. $l\leftarrow k$
2. 将 l 个表两两一组，分成 $l/2$ 组
3. 每组的 2 个表进行归并
4. if　l 为奇数
5. then $l\leftarrow l/2+1$
6. else $l\leftarrow l/2$
7. if　$l>1$ then goto 2
8. return L

最坏情况的时间复杂度：每个表的元素被归并 $\log k$ 次，每次归并比较 1 次. 因此总的时间 $W(n)=O(n\log k)$.

（3）构造决策树如下：

1. 如果算法比较元素 a_i 和 a_j，那么将结点标记为 (i,j)；
2. 结点被标记 (i,j) 之后，

若 $a_i<a_j$，下一步算法比较 a_k 与 a_l，那么 (i,j) 的左儿子标记为 (k,l)；若比较后算法结束，用输入把左儿子标记为树叶.

若 $a_i>a_j$，下一步算法比较 a_k 与 a_l，那么 (i,j) 的右儿子标记为 (k,l)；若比较后算法结束，用输入把右儿子标记为树叶.

将 n 个数分成 k 组，每组 n/k 个数，先选 L_1 中的 n/k 个数，然后从剩下的 $n-n/k$ 个数中选 L_2 中的数，…，从 $n-(k-1)n/k$ 个数中选 n/k 个数. 因此不同的输入个数为 $n!/[(n/k)!]^k$，在决策树中对应了所有的树叶. 给定一个输入，从树根开始，通过比较运算沿着一条路径走到

树叶,那么决策树的深度代表了最坏情况下的时间复杂度,树深

$$d=\left\lceil \log \frac{n!}{[(n/k)!]^k} \right\rceil \geqslant \log(n!) - k\log((n/k)!)$$
$$=\Theta(n\log n - k(n/k)(\log n - \log k)) = \Theta(n\log k)$$

二分归并算法是最优的算法.

8.5 设 n 个点的数组 $X[1..n]$,$X[i]$为第 i 个点的横坐标. 假设算法 $A(X)$是求解直线上最近邻点对的算法,A 的输出是 X 中最近的两个点及其距离 d. 可以设计如下算法求解数的唯一性问题.

算法 Unique

输入:$S=\{a_1, a_2, \cdots, a_n\}$

输出:"Yes"或"No" //若存在 $a_i=a_j$,则输出"No";否则输出"Yes"

1. for i←1 to n do $X[i]=a_i$
2. $A(X)$
3. if d=0 then return "No"
4. else return "Yes"

根据练习 8.8 的结果,唯一性问题的时间复杂度的下界是 $\Theta(n\log n)$. 问题转化为"判断是否存在 i 和 j 使得 $x_i=x_j$ 的"唯一性问题,根据上述分析求解该问题的算法的时间复杂度的下界是 $\Omega(n\log n)$. 可以如下设计算法:先对 X 排序,然后顺序检查相邻元素是否相等. 该算法在最坏情况下的时间复杂度是 $O(n\log n)$. 从而证明了 $\Omega(n\log n)$是一个紧的下界.

8.6 n=4 时,冒泡排序的部分决策树如图 8.1 所示.

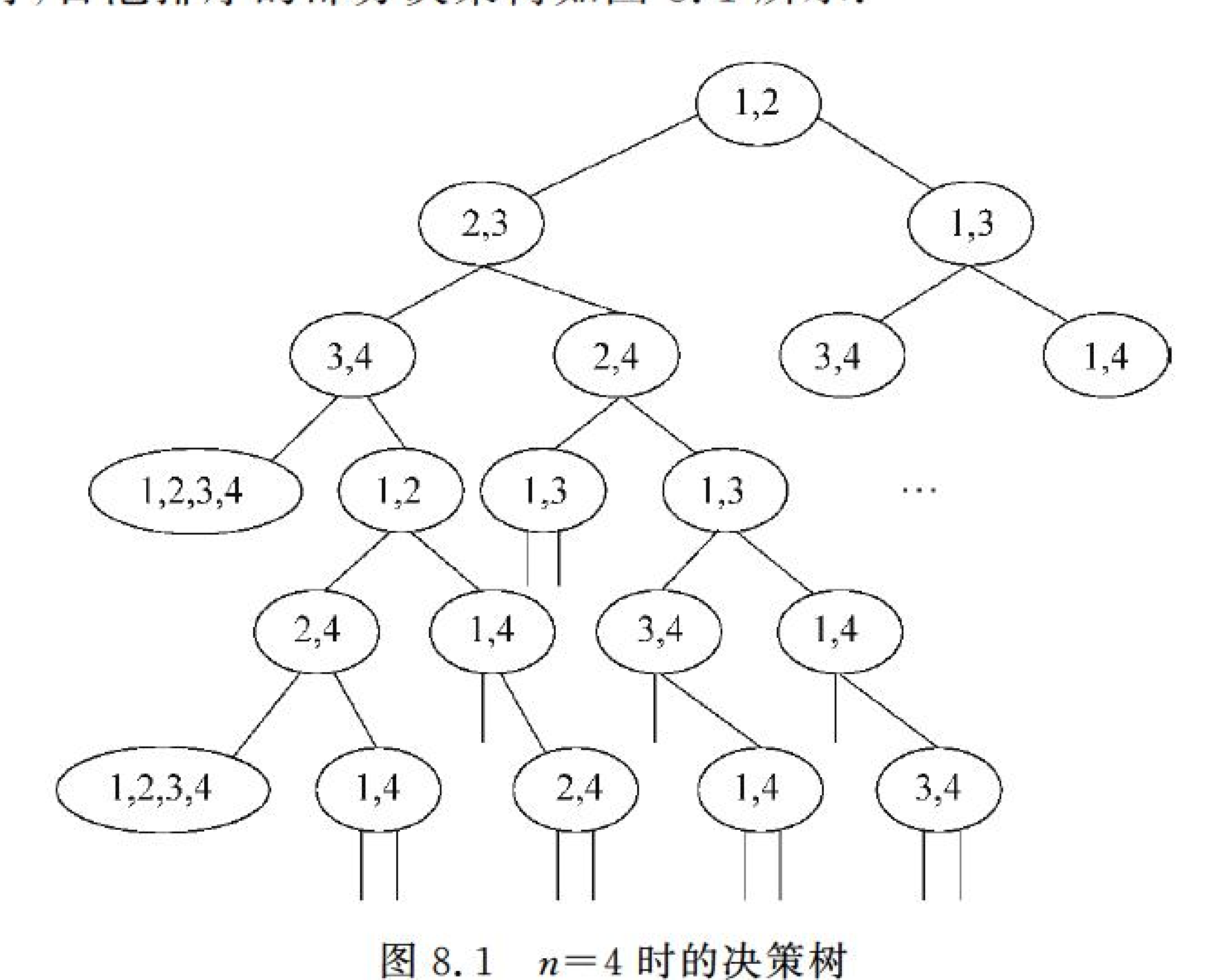


图 8.1 n=4 时的决策树

8.7 (1) 二分查找算法.

Search(A, l, r)

1. if l=r then return l
2. else k←$(l+r)/2$
3. if A[k]=k then return k
4. else if A[k]<k then l←k+1

5.　　　else $r \leftarrow k-1$

6.　　　Search(A, l, r)

初始令 $l=1, r=n$,调用上述算法即可. 算法最坏情况下的时间复杂度为 $O(\log n)$.

(2) 构造决策树,$A[i]$与整数 i 的比较记作结点 i. 如果 $A[i]=i$,算法结束,将 i 标记为树叶并作为 i 的左儿子. 如果 $A[i]\neq i$,将下次比较的标记作为 i 的右儿子. 这棵树有 n 片树叶,树深至少是 $\Omega(\log n)$,因此求解该问题的算法的最坏情况下时间复杂度下界是 $\Omega(\log n)$.

8.8　证　设输入是多重集 $S=\{n_1 \cdot a_1, n_2 \cdot a_2, \cdots, n_k \cdot a_k\}$,$S$ 含有 k 种元素,元素 a_i 的重复度是 n_i,$i=1,2,\cdots,k$,且 $\sum_{i=1}^{k} n_i = n$. 如下构造决策树:

1. 算法第一次比较的元素是 a_i 和 a_j,那么树根标记为(i,j).

2. 如果 $a_i=a_j$,那么算法结束,并将输入标记树叶,作为(i,j)的左儿子.

3. 如果 $a_i\neq a_j$,算法下一步比较的元素是 a_k 和 a_l,那么将(k,l)标记为(i,j)的右儿子;如果比较后算法结束,将(i,j)的右儿子作为树叶,并标记为输入.

多重集 S 的排列数为

$$\binom{n}{n_1 n_2 \cdots n_k} = \frac{n!}{n_1! n_2! \cdots n_k!}$$

由于存在着 n 个元素都不相等的输入,即 $n_1=n_2=\cdots=n_k=1$. 对这个输入,决策树的叶片数达到 $n!$,树的深度为 $\Theta(n\log n)$,从而证明了在最坏情况下算法时间复杂度的下界是 $\Theta(n\log n)$.

显然这是一个紧的界. 先用 $O(n\log n)$时间将 S 中的数按照从小到大的顺序排列为 a_1', $a_2', \cdots, a_n'$,相同的数一定是连续分布的. 然后用 $O(n)$时间顺序检查每对相邻的元素 a_i'和 a_{i+1}'是否相等,总计用 $O(n\log n)$时间就可以判定 S 中的元素是否唯一.

8.9　(1) 在 L 中找第 m 小的数 a_i;然后把 a_i 和其他数比较,找到所有小于 a_i 的数;对所有小于 a_i 的 $m-1$ 个数进行排序;然后从小到大输出. 找 a_i 的时间是 $O(n)$,找所有比 a_i 小的数的时间是 $O(n)$,对 $m-1$ 个数排序时间是 $m\log m$,而 $m=\Theta(n/\log n)$,于是

$$\begin{aligned} T(n) &= O(n) + O(n) + O\left(\frac{n}{\log n}\log\frac{n}{\log n}\right) \\ &= O(n) + O\left(\frac{n}{\log n}(\log n - \log\log n)\right) \\ &= O(n) \end{aligned}$$

(2) **证**　任何算法都对 m 个数排序,时间复杂度 $T(n)=\Omega(m\log m)$,如果 $m=\omega(n/\log n)$,则

$$\begin{aligned} T(n) &= \Omega(m\log m) = \Omega\left(\omega\left(\frac{n}{\log n}\log\frac{n}{\log n}\right)\right) \\ &= \omega\left(\frac{n}{\log n}(\log n - \log\log n)\right) \\ &= \omega\left(n - \frac{n\log\log n}{\log n}\right) = \omega(n) \end{aligned}$$

8.10　证　设第 k 小的数为 t,首先定义决定性的比较与非决定性的比较. 决定性的比

较就是建立了 x 与 t 关系的比较. 例如:

$$\exists y(x > y \text{ 且 } y \geqslant t),\ x \text{ 满足上述条件的第一次比较}$$

$$\exists y(x < y \text{ 且 } y \leqslant t),\ x \text{ 满足上述条件的第一次比较}$$

当 $x>t, y<t$ 时,$x>y$ 的比较不是决定性的. 为找到第 k 小的数 t,必须要做 $n-1$ 次决定性的比较.

设 A 是找第 k 小的任意算法,针对算法 A 如下构造输入.

1. 分配一个值给第 k 小的数 t.

2. 如果 A 比较 x 与 y,且 x 与 y 没有被赋值,那么赋值 x, y 使得 $x>t, y<t$.

3. 如果 A 比较 x 与 y,且 $x>t$,y 没被赋值,则赋值 y 使得 $y<t$.

4. 如果 A 比较 x 与 y,且 $x<t$,y 没被赋值,则赋值 y 使得 $y>t$.

5. 令 $c=\min\{k-1, n-k\}$,如果存在 c 个元素已得到小于 t 的值,则对未赋值的全部分配大于 t 的值.

6. 如果存在 c 个元素已得到大于 t 的值,则对未赋值的全部分配小于 t 的值.

7. 如果剩下 1 个元素则分配 t 给它.

这样赋值的输入使得 A 在这个输入下所进行的上述比较都是非决定性的. 这样的比较至少有 $c=\min\{k-1, n-k\}$个. 因此总比较次数至少为

$$n-1+c = n-1+\min\{k-1, n-k\} = n+\min\{k, n-k+1\}-2$$

8.11 证 考虑 8.8 题的唯一性问题,如下构造本题的最小生成树与唯一性问题的归约.

算法 Unique

输入:$S=\{a_1, a_2, \cdots, a_n\}$

输出:"Yes"或"No"　　//若存在 $a_i=a_j$,则输出"No";否则为"Yes"

1. 把所有的 a_i 变成平面直角坐标系的点$(a_i, 0)$,$i=1,2,\cdots,n$,规定任意两点之间有边.

2. 调用求最小生成树的算法计算 G 的一棵最小生成树,确定树中的最短边的长度 e.

3. 如果 $e=0$,则输出"No";否则输出"Yes".

算法 Unique 的时间复杂度为 $O(n)$+计算最小生成树的时间. 由于求解唯一性问题的算法类的时间复杂度下界是 $\Theta(n\log n)$,因此该算法的时间复杂度是 $\Omega(n\log n)$. 从而证明了计算上述最小生成树的时间复杂度的下界是 $\Omega(n\log n)$.

第 9 章 NP 完全性

9.1 内容提要

1. 基本概念

判定问题与组合优化问题

判定问题 只有两个答案“是”和“否”，或者“Yes”和“No”，可表示成 $\Pi=\langle D_\Pi,Y_\Pi\rangle$，其中 D_Π 是实例集合，$Y_\Pi\subseteq D_\Pi$ 由所有答案为“Yes”的实例组成.

组合优化问题 Π^* 由 3 部分组成：

(1) **实例集** D_{Π^*}.

(2) 对每一个实例 $I\in D_{\Pi^*}$，有一个有穷的非空集合 $S(I)$，$S(I)$的元素称作 I 的**可行解**.

(3) 对每一个可行解 $s\in S(I)$，有一个正整数 $c(s)$，称作 s 的**值**.

当 Π^* 是**最小化问题**（**最大化问题**）时，如果 $s^*\in S(I)$，使得对所有的 $s\in S(I)$，

$$c(s^*)\leqslant c(s)\quad (c(s^*)\geqslant c(s))$$

则称 s^* 是 I 的**最优解**，$c(s^*)$是 I 的**最优值**，记作 $\mathrm{OPT}(I)$.

对应 Π^* 的判定问题 $\Pi=\langle D_\Pi,Y_\Pi\rangle$ 定义如下：$D_\Pi=\{(I,K)\mid I\in D_{\Pi^*},K\in \mathbf{Z}^*\}$，其中 $\mathbf{Z}^*$ 是非负整数集合. 当 Π^* 是最小化问题时，$Y_\Pi=\{(I,K)\mid \mathrm{OPT}(I)\leqslant K\}$；当 Π^* 是最大化问题时，$Y_\Pi=\{(I,K)\mid \mathrm{OPT}(I)\geqslant K\}$.

算法的时间**复杂度**为 $f(n)$：对规模为 n 的任意实例，算法在 $f(n)$步内停止.

多项式时间算法 以多项式为时间复杂度的算法.

易解的问题 有多项式时间算法的问题.

难解的问题 不存在多项式时间算法的问题.

多项式时间验证算法

设判定问题 $\Pi=\langle D,Y\rangle$，如果存在两个输入变量的多项式时间算法 A 和多项式 p，对每一个实例 $I\in D$，$I\in Y$ 当且仅当存在 t，$|t|\leqslant p(|I|)$，且 A 对输入 I 和 t 输出“Yes”，则称 Π 是**多项式时间可验证的**，A 是 Π 的**多项式时间验证算法**，而当 $I\in Y$ 时，称 t 是 $I\in Y$ 的**证据**.

P 类 所有多项式时间可解的判定问题组成的问题类.

NP 类 由所有多项式时间可验证的判定问题组成的问题类.

多项式时间变换

设判定问题 $\Pi_1=\langle D_1,Y_1\rangle$，$\Pi_2=\langle D_2,Y_2\rangle$. 如果函数 f：$D_1\rightarrow D_2$ 满足条件：

(1) f 是多项式时间可计算的,即存在计算 f 的多项式时间算法,

(2) 对所有的 $I \in D_1$,$I \in Y_1 \Leftrightarrow f(I) \in Y_2$,

则称 f 是 Π_1 到 Π_2 的**多项式时间变换**.

如果存在 Π_1 到 Π_2 的多项式时间变换,则称 Π_1 **可多项式时间变换到** Π_2,记作 $\Pi_1 \leqslant_p \Pi_2$.

NP 完全的问题

如果对所有的 $\Pi' \in \mathbf{NP}$,$\Pi' \leqslant_p \Pi$,则称 Π 是 **NP 难的**. 如果 Π 是 NP 难的且 $\Pi \in \mathbf{NP}$,则称 Π 是 **NP 完全的**.

2. 重要结果

定理 9.1 $\mathbf{P} \subseteq \mathbf{NP}$.

定理 9.2 $\leqslant_p$ 具有传递性. 即,设 $\Pi_1 \leqslant_p \Pi_2$,$\Pi_2 \leqslant_p \Pi_3$,则 $\Pi_1 \leqslant_p \Pi_3$.

定理 9.3 设 $\Pi_1 \leqslant_p \Pi_2$,则 $\Pi_2 \in \mathbf{P}$ 蕴涵 $\Pi_1 \in \mathbf{P}$.

推论 9.4 设 $\Pi_1 \leqslant_p \Pi_2$,那么,若 Π_1 是难解的,则 Π_2 也是难解的.

定理 9.5 如果存在 NP 难的问题 $\Pi \in \mathbf{P}$,则 $\mathbf{P} = \mathbf{NP}$.

推论 9.6 假设 $\mathbf{P} \neq \mathbf{NP}$,那么,如果 Π 是 NP 难的,则 $\Pi \notin \mathbf{P}$.

定理 9.7 如果存在 NP 难的问题 Π' 使得 $\Pi' \leqslant_p \Pi$,则 Π 是 NP 难的.

推论 9.8 如果 $\Pi \in \mathbf{NP}$ 并且存在 NP 完全问题 Π' 使得 $\Pi' \leqslant_p \Pi$,则 Π 是 NP 完全的.

下述问题是 NP 完全的:

SAT(Cook-Levin 定理),**MAX-SAT**,**3-SAT**,**顶点覆盖**,**团**,**独立集**,**有向哈密顿回路**,**哈密顿回路**,**货郎问题**,**恰好覆盖**,**子集和**,**0-1 背包**,**双机调度**,**装箱**.

3. 常用技巧

证明Π是 NP 完全的.

(1) 证明$\boldsymbol{\Pi} \in \mathbf{NP}$,通常是给出 Π 的非确定型多项式时间算法.

(2) 找到一个已知的 NP 完全问题 Π',并证明 $\Pi' \leqslant_p \Pi$.

NP 完全性证明的方法

限制法,局部替换法,构件设计法.

9.2 习 题

9.1 写出下述优化问题对应的判定问题:

(1) 最长回路问题:任给无向图 G,求 G 中一条最长的初级(即顶点不重复的)回路.

(2) 多机调度问题:任给有穷的作业集 A 和 m 台相同的机器,作业 a 的处理时间为正整数 $t(a)$,每一项作业可以在任一台机器上完成. 如何把作业分配给机器才能使完成所有作业的时间最短?即,如何把 A 划分成 m 个不相交的子集 A_i,使得

$$\max\left\{\sum_{a \in A_i} t(a) \mid i = 1, 2, \cdots, m\right\}$$

最小?

(3) 图着色问题:任给无向图 $G = \langle V, E \rangle$,给 G 的每一个顶点涂一种颜色,要求任一条边的两个端点的颜色都不相同. 如何用最少的颜色给 G 的顶点着色?即,求映射 $f: V \to$

$\mathbf{Z}^+$ 满足条件 $\forall(u,v)\in E, f(u)\neq f(v)$，且使 $|\{f(u)|u\in V\}|$ 最小.

9.2 给出下述判定问题的简短证据，证明它们属于 **NP**.

（1）**子图同构**：任给两个图 $G=<V,E>$ 和 $H=<V_1,E_1>$，问 H 与 G 的一个子图同构吗？即，存在单射 $f: V_1\to V$ 使得 $\forall u,v\in V_1,(u,v)\in E_1\Rightarrow(f(u),f(v))\in E$？

（2）**度数有界的生成树**：任给一个连通图 G 和正整数 D，问 G 有一棵顶点度数不超过 D 的生成树吗？

9.3 证明在 9.1 题中给出的判定问题属于 **NP**.

9.4 设判定问题 $\Pi=<D,Y>$，称 $\bar{\Pi}=<D,D-Y>$ 为 Π 的**补问题**，即对每一个实例 I，I 在 $\bar{\Pi}$ 中的答案为"Yes"当且仅当 I 在 Π 中的答案为"No".

试给出 HC 的补问题 $\overline{\text{HC}}$，又问：你能给出 $\overline{\text{HC}}$ 的简短证据，并证明 $\overline{\text{HC}}\in\mathbf{NP}$ 吗？为什么？

9.5 给出下述问题之间的多项式时间变换：

（1）**独立集到团.**

（2）**VC 到团.**

（3）**HC 到子图同构.**

（4）**团到 0-1 整数规划**，其中

0-1 整数规划：给定 $m\times n$ 的整数矩阵 $\boldsymbol{A}$，n 维整数列向量 $\boldsymbol{c}$，m 维整数列向量 $\boldsymbol{b}$ 以及整数 D，问：存在 n 维 0-1 列向量 $\boldsymbol{x}$ 使得

$$\boldsymbol{A}\boldsymbol{x}\leqslant\boldsymbol{b} \quad 且 \quad \boldsymbol{c}^{\mathrm{T}}\boldsymbol{x}\geqslant D$$

吗？这里向量 $\boldsymbol{a}\leqslant(\geqslant)\boldsymbol{b}$ 表示 $\boldsymbol{a}$ 的每一个分量 $\leqslant(\geqslant)\boldsymbol{b}$ 的每一个分量.

9.6 设 $\Pi_1\leqslant_p\Pi_2$，证明：

（1）若 $\Pi_2\in\mathbf{P}$，则 $\Pi_1\in\mathbf{P}$.

（2）若 $\Pi_2\in\mathbf{NP}$，则 $\Pi_1\in\mathbf{NP}$.

9.7 证明：如果存在 NP 难的问题 $\Pi\in\mathbf{P}$，则 $\mathbf{P}=\mathbf{NP}$.（定理 9.5）

9.8 证明：假设 $\mathbf{P}\neq\mathbf{NP}$，那么，如果 Π 是 NP 难的，则 $\Pi\notin\mathbf{P}$.（推论 9.6）

9.9 证明：如果存在 NP 难的问题 Π' 使得 $\Pi'\leqslant_p\Pi$，则 Π 是 NP 难的.（定理 9.7）

9.10 证明：如果 $\Pi\in\mathbf{NP}$ 并且存在 NP 完全问题 Π' 使得 $\Pi'\leqslant_p\Pi$，则 Π 是 NP 完全的.（定理 9.8）

9.11 **有向哈密顿通路（有向 HP）**：给定有向图 D，经过 D 中每一个顶点恰好一次的通路称作哈密顿通路. 问：D 中有一条哈密顿通路吗？

仿照教材定理 9.16 证明中 **3SAT** 到**有向 HC** 的多项式时间变换构造 **3SAT** 到**有向 HP** 的多项式时间变换.

9.12 构造**有向 HC** 到**有向 HP** 的多项式时间变换.

9.13 构造**有向 HP** 到 **HP** 的多项式时间变换. **哈密顿通路（HP）**是**有向 HP** 的无向图形式.

证明题 9.14～9.20 中叙述的问题是 NP 完全的.

9.14 **划分**：任给 n 个正整数 $a_1,a_2,\cdots,a_n$，问：能把这 n 个数分成和相等的两部分吗？即，存在子集 $T\subseteq I=\{1,2,\cdots,n\}$，使得

$$\sum_{i\in T}a_i = \sum_{i\in I-T}a_i$$

吗?

9.15　子图同构.(定义见题9.2)

9.16　度数有界的生成树.(定义见题9.2)

9.17　最长通路:任给无向图 $G=\langle V,E\rangle$ 和正整数 $K\leqslant|V|$,问:G 中存在边数不少于 K 的初级(即,顶点不重复的)通路吗?

9.18　反馈顶点集:任给有向图 $D=\langle V,E\rangle$ 和正整数 $K\leqslant|V|$,问:存在子集 $V'\subseteq V$ 使得 $|V'|\leqslant K$,并且 D 中的每一条回路上至少有一个 V' 的顶点吗?

9.19　支配集:任给无向图 $G=\langle V,E\rangle$ 和正整数 $K\leqslant|V|$,问:存在子集 $V'\subseteq V$ 使得 $|V'|\leqslant K$,并且 $V-V'$ 中的每一个顶点都至少与 V' 中的一个顶点相邻吗?

9.20　可着三色:任给一个图 $G=\langle V,E\rangle$,问:G 是可着三色的吗?即,存在函数 f:$V\to\{1,2,3\}$ 使得 $\forall(u,v)\in E$ 且 $u\neq v$,有 $f(u)\neq f(v)$ 吗?

9.3　习题解答与分析

9.1　解　(1) **最长回路**:任给无向图 $G=\langle V,E\rangle$ 和正整数 $L\leqslant|V|$,问:G 中有长度不小于 L 的初级(即顶点不重复的)回路吗?

(2) **多机调度**:任给有穷的作业集 A,m 台相同的机器和正整数 D,作业 a 的处理时间为正整数 $t(a)$,每一项作业可以在任一台机器上完成. 问:能把作业分配给机器使完成所有作业的时间不超过 D 吗?即,能把 A 划分成 m 个不相交的子集 $A_i(i=1,2,\cdots,m)$,使得

$$\max\left\{\sum_{a\in A_i}t(a)\ \middle|\ i=1,2,\cdots,m\right\}\leqslant D$$

吗?

(3) **图着色**:任给无向图 $G=\langle V,E\rangle$ 和正整数 K,问:能用不超过 K 种颜色给 G 的每一个顶点涂一种颜色,要求任一条边的两个端点的颜色都不相同吗?即,存在映射 f:$V\to\mathbf{Z}^+$ 满足条件 $\forall(u,v)\in E$,$f(u)\neq f(v)$,且 $|\{f(u)|u\in V\}|\leqslant K$ 吗?

9.2　解　(1) 简短证据是单射 f:$V_1\to V$.

子图同构的非确定型多项式时间算法:猜想一个单射 f:$V_1\to V$,检查是否 $\forall u,v\in V_1$,$(u,v)\in E_1\Rightarrow(f(u),f(v))\in E$. 若是,则回答"yes",否则回答"no".

(2) 简短证据是 G 的 $n-1$ 条边,其中 n 是 G 的顶点数.

度数有界的生成树的非确定型多项式时间算法:猜想 $n-1$ 条边 E_1,检查 E_1 是否构成 G 的连通子图且每一个顶点与 E_1 中关联的边数是否都大于等于1且小于等于 D. 若是,则回答"yes",否则回答"no".

9.3　证　(1) **最长回路**的非确定型多项式时间算法:猜想 G 的一条初级回路 C,若 C 的长度大于等于 L,则回答"yes",否则回答"no".

(2) **多机调度**的非确定型多项式时间算法:猜想一个作业的分配方案,计算分配给每一台机器的处理时间的和,若 m 台机器的处理时间都不超过 D,则回答"yes",否则回答"no". 即,猜想 A 的一个划分 $A_1,A_2,\cdots,A_m$,若 $\max\left\{\sum_{a\in A_i}t(a)\ \middle|\ i=1,2,\cdots,m\right\}\leqslant D$,则回

答“yes”，否则回答“no”.

(3) **图着色**的非确定型多项式时间算法：猜想用不超过 K 种颜色给 G 的每一个顶点涂一种颜色，检查每一条边的两个端点的颜色是否都不相同. 若是，则回答“yes”，否则回答“no”. 即，猜想一个单射 $f: V\to\{1,2,\cdots,K\}$，若满足条件 $\forall(u,v)\in E, f(u)\neq f(v)$，则回答“yes”，否则回答“no”.

9.4 解 HC 的补问题 $\overline{\text{HC}}$：任给无向图 $G=\langle V,E\rangle$，问：G 中不存在哈密顿回路吗？

不能给出它的简短证据，也不能证明它属于 **NP**. 要证明 G 不是哈密顿图，就要检查 G 的 n 个顶点的每一种可能的排列是否是一条哈密顿回路. 若都不是，则 G 不是哈密顿图，回答“yes”. 现在还不知道在本质上更短的“证据”. 固定第一个顶点且在 2 个相反的排列中只取一个，共有 $(n-1)!/2$ 个排列. 列举所有这些排列的字符串的长度是 n 的指数函数.

9.5 解 (1) **独立集**到**团**的多项式时间变换 f. 对**独立集**每一个的实例 I：无向图 $G=\langle V,E\rangle$ 和非负整数 $J\leqslant|V|$，**团**对应的实例 $f(I)$：无向图 $\bar{G}=\langle V,\bar{E}\rangle$ 和 J，其中 $\bar{E}=\{(u,v)\mid u,v\in V, u\neq v$ 且 $(u,v)\notin E\}$.

显然，f 是多项式时间可计算的. 设 $V'\subseteq V$，V' 是 G 的独立集 $\Leftrightarrow V'$ 是 $\bar{G}$ 的团. 从而，G 有顶点数不少于 J 的独立集 $\Leftrightarrow\bar{G}$ 有顶点数不少于 J 的团，即 I 是是实例 $\Leftrightarrow f(I)$ 是是实例. 故 f 是**独立集**到**团**的多项式时间变换.

(2) **VC** 到**团**的多项式时间变换 f. 对 **VC** 的每一个实例 I：无向图 $G=\langle V,E\rangle$ 和非负整数 $K\leqslant|V|$，**团**对应的实例 $f(I)$：无向图 $\bar{G}=\langle V,\bar{E}\rangle$ 和 $J=|V|-K$.

(3) **HC** 到**子图同构**的多项式时间变换 f. 对 **HC** 的每一个实例 I：无向图 $G=\langle V,E\rangle$，**子图同构**对应的实例 $f(I)$：无向图 $G=\langle V,E\rangle$ 和 $H=\langle V_1,E_1\rangle$，其中 $V_1=V$，E_1 是一个经过 V 的所有顶点的圈上的边. 即，设 $V=\{v_1,v_2,\cdots,v_n\}$，取

$$E_1=\{(v_i,v_{i+1})\mid i=1,2,\cdots,n-1\}\cup\{(v_n,v_1)\}$$

(4) **团**到 **0-1 整数规划**的多项式时间变换 f. 对**团**的每一个实例 I：无向图 $G=\langle V,E\rangle$ 和非负整数 $J\leqslant|V|$，其中 $V=\{v_1,v_2,\cdots,v_n\}$，**0-1 整数规划**对应的实例 $f(I)$：

$$\begin{aligned}&\sum_{i=1}^{n}x_i\geqslant J\\&x_i+x_j\leqslant 1,\quad (v_i,v_j)\notin E, i\neq j\\&x_i=0,1,\quad i=1,2,\cdots,n\end{aligned}$$

显然，f 是多项式时间可计算的. 设 $V'\subseteq V$ 是 G 的一个团且 $|V'|\geqslant J$，令

$$x_i=\begin{cases}1, & 若\ v_i\in V'\\0, & 否则\end{cases}$$

则当 $(v_i,v_j)\notin E$ 时，$x_i+x_j\leqslant 1$，且 $\sum_{i=1}^{n}x_i=|V'|\geqslant J$. 反之，取 $V'=\{v_i\mid x_i=1\}$，则 $\forall v_i, v_j\in V'$，$x_i+x_j=2$，从而 $(v_i,v_j)\in E$. 又 $|V'|=\sum_{i=1}^{n}x_i\geqslant J$，即 $V'\subseteq V$ 是 G 的一个顶点数不少于 J 的团.

9.6 证 (1) 设 f 是 Π_1 到 Π_2 的多项式时间变换，A 是 Π_2 的多项式时间算法，f 和 A 的时间复杂度上界分别为多项式 p 和 q，且不妨设 q 是单调递增的. 构造 Π_1 的算法 B 如

下：对 Π_1 的每一个实例 I，首先计算 $f(I)$，然后对 $f(I)$应用算法 A. B 对 I 回答“yes”$\Leftrightarrow$A 对 $f(I)$回答“yes”.

显然，B 的回答总是正确的. $f(I)$的计算时间是 $p(|I|)$，A 对 $f(I)$的运行时间是 $q(|f(I)|)$，B 的总运行时间是 $p(|I|)+q(|f(I)|)$. 注意到$|f(I)|\leqslant kp(|I|)$，k 是一个常数，因而 $p(|I|)+q(|f(I)|)\leqslant p(|I|)+q(kp(|I|))$，这是$|I|$的多项式，故 B 是多项式时间的.

(2) 设 f 是Π_1 到 Π_2 的多项式时间变换，A 是 Π_2 的多项式时间验证算法，f 和 A 的时间复杂度上界分别为多项式 p 和q，且不妨设 q 是单调递增的. 构造 Π_1 的验证算法 B 如下：对 Π_1 的每一个实例 I，首先计算 $f(I)$，然后对 $f(I)$和“猜想”t 应用 A，B 对 I 和t 回答“yes”$\Leftrightarrow$A 对 $f(I)$和 t 回答“yes”.

因为 $\Pi_2\in \mathbf{NP}$，$f(I)$是是实例$\Leftrightarrow$存在 t，$|t|\leqslant r(|f(I)|)$，使得 A 对 $f(I)$和 t 回答“yes”，其中 r 是一个多项式，不妨设 r 是单调增加的. 于是，I 是是实例$\Leftrightarrow f(I)$是是实例$\Leftrightarrow$存在 t，$|t|\leqslant r(|f(I)|)$，使得 A 对 $f(I)$和 t 回答“yes”$\Leftrightarrow$存在 t，$|t|\leqslant r(|f(I)|)$，使得 B 对 I 和 t 回答“yes”. 而$|f(I)|\leqslant kp(|I|)$，故 $r(|f(I)|)\leqslant r(kp(|I|))$，后者是$|I|$的多项式. 又类似(1)可证 B 是多项式时间的，从而得证 $\Pi_1\in \mathbf{NP}$.

9.7 证 $\forall \Pi'\in \mathbf{NP}$，因为 Π 是 NP 难的，有 $\Pi'\leqslant_p \Pi$. 而 $\Pi\in \mathbf{P}$，由定理 9.3，有 $\Pi'\in \mathbf{P}$. 得证 $\mathbf{P}=\mathbf{NP}$.

9.8 证 假设不然，存在 NP 难的 Π，使得 $\Pi\in \mathbf{P}$，由定理 9.5，有 $\mathbf{P}=\mathbf{NP}$. 与假设 $\mathbf{P}\neq \mathbf{NP}$ 矛盾.

9.9 证 $\forall \Pi''\in \mathbf{NP}$，因为 Π'是 NP 难的，有 $\Pi''\leqslant_p \Pi'$. 由$\leqslant_p$ 的传递性(定理 9.2)，有 $\Pi''\leqslant_p \Pi$. 得证 Π 是 NP 难的.

9.10 证 因为存在 NP 完全问题 Π' 使得 $\Pi'\leqslant_p \Pi$，而 NP 完全问题也是 NP 难的，由定理 9.7，Π 是 NP 难的. 又已知 $\Pi\in \mathbf{NP}$，故 Π 是 NP 完全的.

9.11 解 只需删去教材中图 9.3 中的边$<s_n,s_0>$.

9.12 解 任给有向图 $D=<V,E>$，不妨设 G 中没有环；否则删去所有的环，这不影响图是否是哈密顿图. 任取一个顶点 u，把 u 替换成 2 个新顶点 s 和 t，把以 u 为终点的每一条边$<v,u>$替换成$<v,t>$，以 u 为起点的每一条边$<u,v>$替换成$<s,v>$. 记所得到的图为 $D'=<V'.E'>$. 即

$$V' = (V-\{u\})\cup\{s,t\}\quad s,t\notin V$$

$$E' = (E-\{<v,u>,<u,v>|\ v\in V\})\cup\{<v,t>|\ v\neq u,<v,u>\in E\}\cup \{<s,v>|\ v\neq u,<u,v>\in E\}$$

显然，D'是多项式时间可构造的且 D'中任何哈密顿通路(如果有)都是以 s 为始点、以 t 为终点. 又不难看出，D 有哈密顿回路$\Leftrightarrow D'$有从 s 到 t 的哈密顿通路，因此这是**有向 HC** 到**有向 HP** 的多项式时间变换.

9.13 解 采用教材定理 9.17 证明中多项式时间变换. 任给有向图 $D=<V,E>$，把 D 的每一个顶点 v 替换成 3 个顶点 v^{in}、v^{mid}和 v^{out}，用边连接 v^{in}和 v^{mid}、v^{mid}和 v^{out}. D 中的每一条有向边$<u,v>$在 G 中替换成$(u^{\text{out}},v^{\text{in}})$. 即

$$V' = \{v^{\text{in}},v^{\text{mid}},v^{\text{out}}\ |\ v\in V\}$$

$$E' = \{(u^{\text{out}},v^{\text{in}})\ |<u,v>\in E\}\cup\{(v^{\text{in}},v^{\text{mid}}),(v^{\text{mid}},v^{\text{out}})\ |\ v\in V\}$$

显然，G 可以在多项式时间内完成构造. 要证 D 有哈密顿通路$\Leftrightarrow G$ 有哈密顿通路. 根据 D 中的哈密顿通路很容易构造出 G 中的哈密顿通路；反之，设 P 是 G 中的一条哈密顿通路. 由于(v^{in}, v^{mid})和(v^{mid}, v^{out})是 G 中与 v^{min} 关联的仅有的两条边，故所有这样的边都在 P 上. 又注意到对不同的 u 和 v，只有 u^{out} 与 v^{in}，u^{in} 与 v^{out} 之间可能有边，故 P 一定是按照 u^{out}, u^{mid}, u^{in} 的顺序经过每一个顶点或者按照 u^{in}, u^{mid}, u^{out} 的顺序经过每一个顶点，P 的两个端点一定是一个 s^{out} 和一个 t^{in}，其中 $s \neq t$. 于是，容易根据 P 构造出 D 中从 s 到 t 的哈密顿通路.

9.14 证 不难证明**划分**$\in$**NP**. 要证**子集和**$\leqslant_p$ **划分**. 可以把**划分**看作**双机调度**的特殊情况——取 $D = \left\lfloor \frac{1}{2}\sum_{i=1}^{n} t_i \right\rfloor$. 检查教材定理 9.18 证明中**子集和**到**双机调度**的多项式时间变换，对每一个**子集和**的实例，对应的**双机调度**实例中的截止时间 D 都恰好等于所有作业的加工时间之和的二分之一，因此这个变换实际上也是**子集和**到**划分**的多项式时间变换.

9.15 证 **子图同构**的非确定型多项式时间算法如下：对任给的两个图 $G=\langle V,E\rangle$ 和 $H=\langle V_1,E_1\rangle$，猜想一个单射 $f: V_1 \to V$，检查 f 是否是 H 到 $f(H)$ 的导出子图的同构映射，即是否 $\forall u,v \in V_1, (u,v)\in E_1 \Rightarrow (f(u),f(v)) \in E$. 若是则回答“yes”，否则回答“no”. 故**子图同构**$\in$**NP**.

上面题 9.5(3)已经给出 **HC** 到**子图同构**的多项式时间变换，得证**子图同构**是 NP 完全的.

9.16 证 题 9.2(2)已经给出**度数有界的生成树**的非确定型多项式时间算法，故属于 **NP**. **HP** 是**度数有界的生成树**的子问题——限制度数的界限 $D=2$. **HP** 到**度数有界的生成树**的多项式时间变换如下：任给无向图 $G=\langle V,E\rangle$，对应的**度数有界的生成树**实例由图 $G=\langle V,E\rangle$ 和度数界限 $D=2$ 组成.

9.17 证 显然**最长通路**$\in$**NP**. **HP** 是**最长通路**的子问题——限制初级通路的长度不小于 $K=n-1$，其中 n 是顶点数. **HP** 到**最长通路**的多项式时间变换如下：任给无向图 $G=\langle V,E\rangle$，对应的**最长通路**实例由图 $G=\langle V,E\rangle$ 和初级通路的边数下界 $K=|V|-1$ 组成.

9.18 证 称满足题中条件的子集 V' 是 D 的反馈顶点集. 显然，V' 是 D 的反馈顶点集$\Leftrightarrow$在 $V-V'$ 的导出子图中没有回路. **反馈顶点集**的验证算法如下：猜想一个顶点数不超过 K 的子集 $V' \subseteq V$，删去 V' 中的顶点及其关联的边，检查在剩余的图中是否有回路. 若没有回路，则输出“yes”；否则输出“no”. 可以在多项式时间内判断一个有向图是否有回路，故上述验证算法是多项式时间的. 得证**反馈顶点集**$\in$**NP**.

要证 **VC**$\leqslant_p$ **反馈顶点集**. 任给无向图 $G=\langle V,E\rangle$ 和 $K\leqslant|V|$，对应的**反馈顶点集**实例由有向图 $D=\langle V',A\rangle$ 和正整数 K 构成，其中 $V'=V\cup E$，$A=\sum_{e\in E} A_e$. A_e 构造如下：将 V 中的顶点任意地排列编号为 $v_1, v_2, \cdots, v_n$. 对每一条边 $e=(v_i,v_j)$，$i<j$，$A_e=\{\langle v_i,v_j\rangle, \langle v_j,e\rangle, \langle e,v_i\rangle\}$. 显然，$D$ 是多项式时间可构造的. 图 9.1 给出了一个简单的

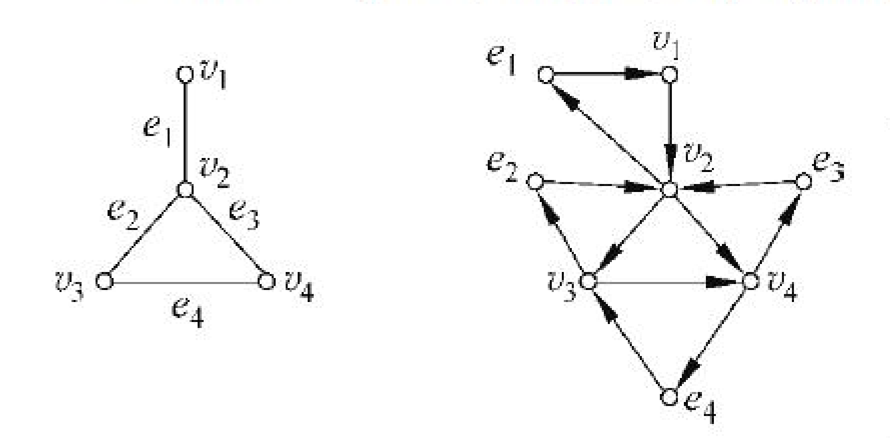


图 9.1 一个变换实例

例子.

在 D 中,对每一个 $e\in E$,由 A_e 的 3 条边构成的三角形是唯一经过 e 的初级回路. 而且任何回路都是由若干个这样的三角形组成的. D 中不存在只经过 V 中顶点的回路.

设 $V^*\subseteq V$ 是 G 的一个顶点覆盖,对 D 中的一条回路,设它经过 $e=(u,v)$,从而也经过 u 和 v. 而 V^* 是 G 的顶点覆盖,必有 $u\in V^*$ 或 $v\in V^*$,故这条回路至少经过 V^* 中的一个顶点. 得证 V^* 是 D 的反馈顶点集.

反之,设 $V^*\subseteq V'$ 是 D 的反馈顶点集. 如果 $V^*\cap E\neq\varnothing$,设 $e=(u,v)\in V^*$. 令 $V_1^*=(V^*-\{e\})\cup\{u\}$,$D$ 中所有经过 e 的回路都经过 u,从而 V_1^* 也是 D 的反馈顶点集且比 V^* 少一个 E 中的顶点. 重复这个做法,可以得到 D 的一个不含 E 中顶点的反馈顶点集. 于是,可以设 $V^*\subseteq V$. 对每一条边 $e=(u,v)\in E$,在 D 中由 A_e 的 3 条边构成的回路上至少有 V^* 中的一个顶点,即 $u\in V^*$ 或 $v\in V^*$,从而在 G 中 V^* 覆盖边 e. 得证 V^* 也是 G 的顶点覆盖.

9.19　证　显然,**支配集**$\in$**NP**. 称满足题中条件的子集 V' 是 G 的一个支配集. 要证 **VC**$\leqslant_p$ **支配集**. 任给 **VC** 的一个实例,它由无向图 $G=\langle V,E\rangle$ 和非负整数 $K\leqslant|V|$ 组成,对应的**支配集**实例由 $G'=\langle V',E'\rangle$ 和正整数 K 构成,其中 G' 由两部分构成:一部分是以 V 为顶点集的完全图;另一部分与 G 的边相关,对每一条边 $e=(u,v)$,有一个以 u、v 和 e 为顶点的三角形. 即 $V'=V\cup E$,

$$E'=\{(u,v)\mid u,v\in V,u\neq v\}\cup\{(u,e),(v,e)\mid e=(u,v)\in E\}$$

显然,G' 可以在多项式时间内构成.

设 $V^*\subseteq V$ 是 G 的一个顶点覆盖,显然 V^* 是 G' 的一个支配集. 反之,设 $V^*\subseteq V'$ 是 G' 的一个支配集. 如果 $V^*\cap E\neq\varnothing$,设 $e=(u,v)\in V^*$. 令 $V_1^*=(V^*-\{e\})\cup\{u\}$,注意,在 G' 中,V 中的任意两个顶点都相邻,而 E 中的任意两个顶点都不相邻,故 V_1^* 仍是 G' 的支配集,但它少了一个 E 中的顶点. 如此重复,可以得到 G' 的一个不含 E 中顶点的支配集,仍把它记作 V^*,即 $V^*\subseteq V$. 显然,对每一条边 $e=(u,v)\in E$,有 $u\in V^*$ 或 $v\in V^*$,从而 V^* 是 G 的一个顶点覆盖.

9.20　证　显然,**可着三色**$\in$**NP**. 要证 **3SAT**$\leqslant_p$ **可着三色**. 任给一个三元合取范式 $F=C_1\wedge C_2\wedge\cdots\wedge C_m$,它有 n 个变元 $x_1,x_2,\cdots,x_n$ 和 m 个简单析取式 $C_j=z_{j1}\vee z_{j2}\vee z_{j3}$,其中 z_{jt} 等于某个 x_i 或 x_i,$j=1,2,\cdots,m$,$t=1,2,3$. 要构造一个图 $G=(V,E)$,使得 F 是可满足的当且仅当 G 是可着三色的.

G 有一个三角形,由 3 个顶点 v_0,v_1,v_2 和连接它们的边组成,这 3 个顶点必须着 3 种颜色,不妨设分别着颜色 0,1,2. 对每一个顶点 x_i,G 有 2 个顶点 x_i 和 $\bar{x}_i$,它们之间有一条边并且都与 v_2 相连,如图 9.2 所示. 把 G 的这个子图记作 G_0. 每一对顶点 x_i 和 $\bar{x}_i$ 只能是一个着颜色 1、另一个着颜色 0,这恰好对应变元 x_i 的取值. 顶点 x_i 着颜色 1 对应变元 x_i 取值 1,顶点 x_i 着颜色 0 对应变元 x_i 取值 0. 因此,G_0 的一个 3-着色恰好对应对变元 $x_1,x_2,\cdots,x_n$ 的一个赋值.

为了将着色与 F 的可满足性联系起来,再对每一个简单析取式 $C_j=z_{j1}\vee z_{j2}\vee z_{j3}$ 构造如图 9.3 所示的子图 G_j. 在这里,若 $z_{jt}=x_i$,则顶点 $z_{jt}=x_i$;若 $z_{jt}=x_i$,则顶点 $z_{jt}=\bar{x}_i$. 可以看出,能够把 G_0 的 3-着色扩大到 G_j 上的 3-着色当且仅当 z_{j1},z_{j2},z_{j3} 中至少有一个着颜

色 1,亦即对变元 $x_1,x_2,\cdots,x_n$ 对应 G_0 的 3-着色的赋值满足 C_j.

图 G 由 G_0 和 $G_1,G_2,\cdots,G_m$ 组成. 根据上面的叙述,可以证明 F 是可满足的当且仅当 G 是可着三色的. 又,由 F 构造 G 显然是可以在多项式时间内完成的. 因此,得证 3SAT 可多项式时间变换到可着三色.

图 9.2 子图 G_0 的结构

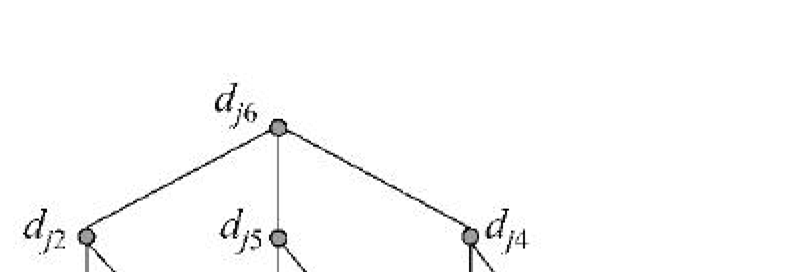
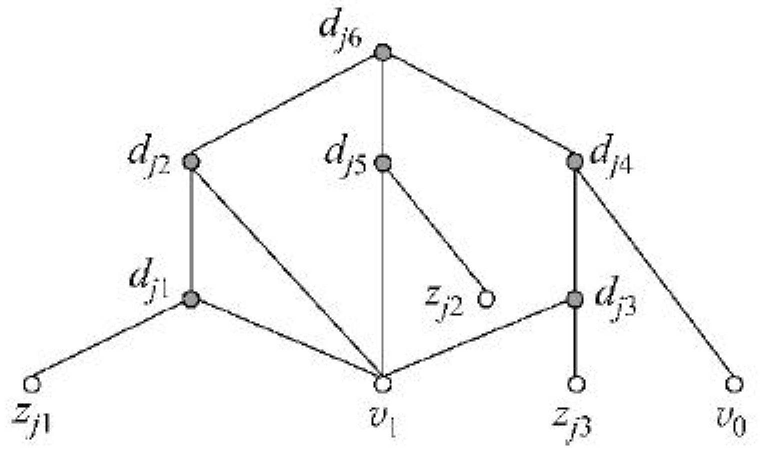



图 9.3 子图 G_j 的结构

第 10 章 近似算法

10.1 内容提要

1. 基本概念

近似算法

如果对组合优化问题 Π 的每一个实例 I，多项式时间算法 A 输出 I 的一个可行解 σ，则称 A 是 Π 的**近似算法**. 记 $A(I)=c(\sigma)$，其中 $c(\sigma)$ 是 σ 的值. 如果 A 总是输出 I 的最优解，则称 A 是 Π 的**最优化算法**.

近似比

当 Π 是最小化问题时，记

$$r_{\mathrm{A}}(I)=\frac{\mathrm{A}(I)}{\mathrm{OPT}(I)}$$

当 Π 是最大化问题时，记

$$r_{\mathrm{A}}(I)=\frac{\mathrm{OPT}(I)}{\mathrm{A}(I)}$$

如果对 Π 的每一个实例 I，$r_{\mathrm{A}}(I)\leqslant r$，则称近似算法 A 的**近似比**为 r，又称 A 是 ***r*-近似算法**. 当 r 是一个常数时，称 **A 具有常数比**.

紧实例 使算法产生的解值与最优值之比等于或可以任意接近近似比的实例.

NP 难的组合优化问题按可近似性的分类：

(1) **完全可近似的**：对任意小的 $\varepsilon>0$，存在 $(1+\varepsilon)$-近似算法.

(2) **可近似的**：存在具有常数比的近似算法.

(3) **不可近似的**：不存在具有常数比的近似算法，除非 **P＝NP**.

多项式时间近似方案 以 $\varepsilon>0$ 和问题的实例 I 作为输入，且对每一个固定的 $\varepsilon>0$ 是 $(1+\varepsilon)$-近似算法.

完全多项式时间近似方案 以 $\varepsilon>0$ 和问题的实例 I 作为输入，以某个二元多项式 $p(|I|,1/\varepsilon)$ 为时间复杂度上界，且对每一个固定的 $\varepsilon>0$，近似比为 $1+\varepsilon$ 的近似算法.

2. 典型的近似算法

最小顶点覆盖问题的算法 MVC.

多机调度的贪心法(G-MPS)，递降贪心法(DG-MPS).

货郎问题的最邻近法(NN)，最小生成树法(MST)，最小权匹配法(MM).

背包问题的贪心算法(G-KK)，多项式近似方案(PTAS)，完全多项式近似方案(FPTAS).

10.2 习　　题

10.1　教材10.1节最小顶点覆盖问题的近似算法MVC任取一条边，把这条边的两个端点加入顶点覆盖集，现在改为只把这条边的一个端点加入顶点覆盖集，其余不变. 试分析这个修改后的算法的近似性能.

10.2　给出"对货郎问题所有满足三角不等式的实例 I，$\mathrm{MM}(I) < \frac{3}{2}\mathrm{OPT}(I)$"（定理10.5）的紧实例.

10.3　装箱问题（优化形式）：任给 n 件物品，物品 j 的重量为 w_j，$1 \leqslant j \leqslant n$. 限制每只箱子装入物品的总重量不超过 B，这里 w_j 和 B 都是正整数，且 $w_j \leqslant B$，$1 \leqslant j \leqslant n$. 要求用最少的箱子装入所有的物品，怎么装法？

考虑下述简单的装法.

首次适合算法(FF)：按照输入顺序装物品，对每一件物品，依次检查每一只箱子，只要能装得下就把它装入. 只有在所有已经打开的箱子都装不下这件物品时，才新打开一只箱子.

证明：对装箱问题的所有实例 I

$$\mathrm{FF}(I) < 2\mathrm{OPT}(I)$$

10.4　证明：装箱问题不存在近似比 $r < \frac{3}{2}$ 的多项式时间近似算法，除非 $\mathbf{P}=\mathbf{NP}$.

10.5　设无向图 $G=\langle V,E\rangle$，$V_1 \cup V_2 = V$，$V_1 \cap V_2 = \varnothing$，称 $(V_1,V_2)=\{(u,v) \mid (u,v) \in E$，且 $u \in V_1, v \in V_2\}$ 是 G 的割集. (V_1,V_2) 中的边称作割边，不在 (V_1,V_2) 中的边称作非割边.

最大割集问题：任给无向图 $G=\langle V,E\rangle$，求 G 的边数最多的割集.

考虑下述最大割集问题的局部改进算法.

算法MCUT：令 $V_1=V$，$V_2=\varnothing$. 如果存在顶点 u，在 u 关联的边中非割边多于割边，如果 $u \in V_1$，则把 u 移到 V_2 中；如果 $u \in V_2$，则把 u 移到 V_1 中. 直到不存在这样的顶点为止，取此时得到的 (V_1,V_2) 作为解.

证明：对求最大割集问题的每一个实例 I

$$\mathrm{OPT}(I) \leqslant 2\mathrm{MCUT}(I)$$

10.6　双机调度问题（优化形式）：有2台相同的机器和 n 项作业 $J_1, J_2, \cdots, J_n$，每一项作业可以在任一台机器上处理，没有顺序的限制，作业 J_i 的处理时间为正整数 t_i，$1 \leqslant i \leqslant n$. 要求把 n 项作业分配给这2台机器使得完成时间最短，即把 $\{1,2,\cdots,n\}$ 划分成 I_1 和 I_2，使得

$$\max\left\{\sum_{i \in I_1} t_i, \sum_{i \in I_2} t_i\right\}$$

最小.

令 $D=\left\lfloor \frac{1}{2}\sum_{i=1}^{n} t_i \right\rfloor$，$B(i)=\left\{t \mid t=\sum_{j \in S} t_j \leqslant D, S \subseteq \{1,2,\cdots,i\}\right\}$，$0 \leqslant i \leqslant n$. $B(i)$ 包括所有前 i 项作业中任意项（可以是0项）作业的处理时间之和，只要这个和不超过所有作业处

理时间之和的二分之一.

试给出关于 $B(i)$ 的递推公式,并利用这个递推公式设计双机调度问题的伪多项式时间算法,进而设计这个问题的完全多项式时间近似方案.

10.3 习题解答与分析

10.1 解 设顶点数为 n,显然,$A(I)\leqslant n-1$,$\mathrm{OPT}(I)\geqslant 1$,$r\leqslant n-1$.

给定紧实例:$G=\langle V,E\rangle$,$V=\{1,2,\cdots,n\}$,$E=\{(i,n)\mid 1\leqslant i\leqslant n-1\}$,如图10.1所示.

算法依次选择 $1,2,\cdots,n-1$. $A(I)=n-1$. 显然,$\mathrm{OPT}(I)=1$,得

$$\frac{\mathrm{A}(I)}{\mathrm{OPT}(I)}=n-1$$

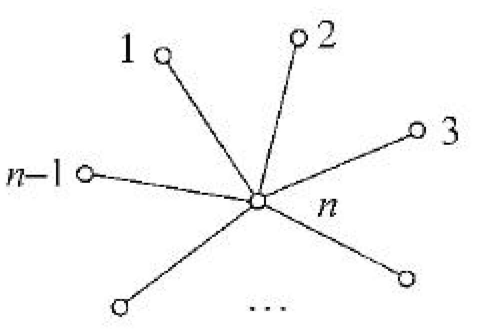


图10.1 一个紧实例

10.2 解 紧实例及MM算法的应用如图10.2所示.

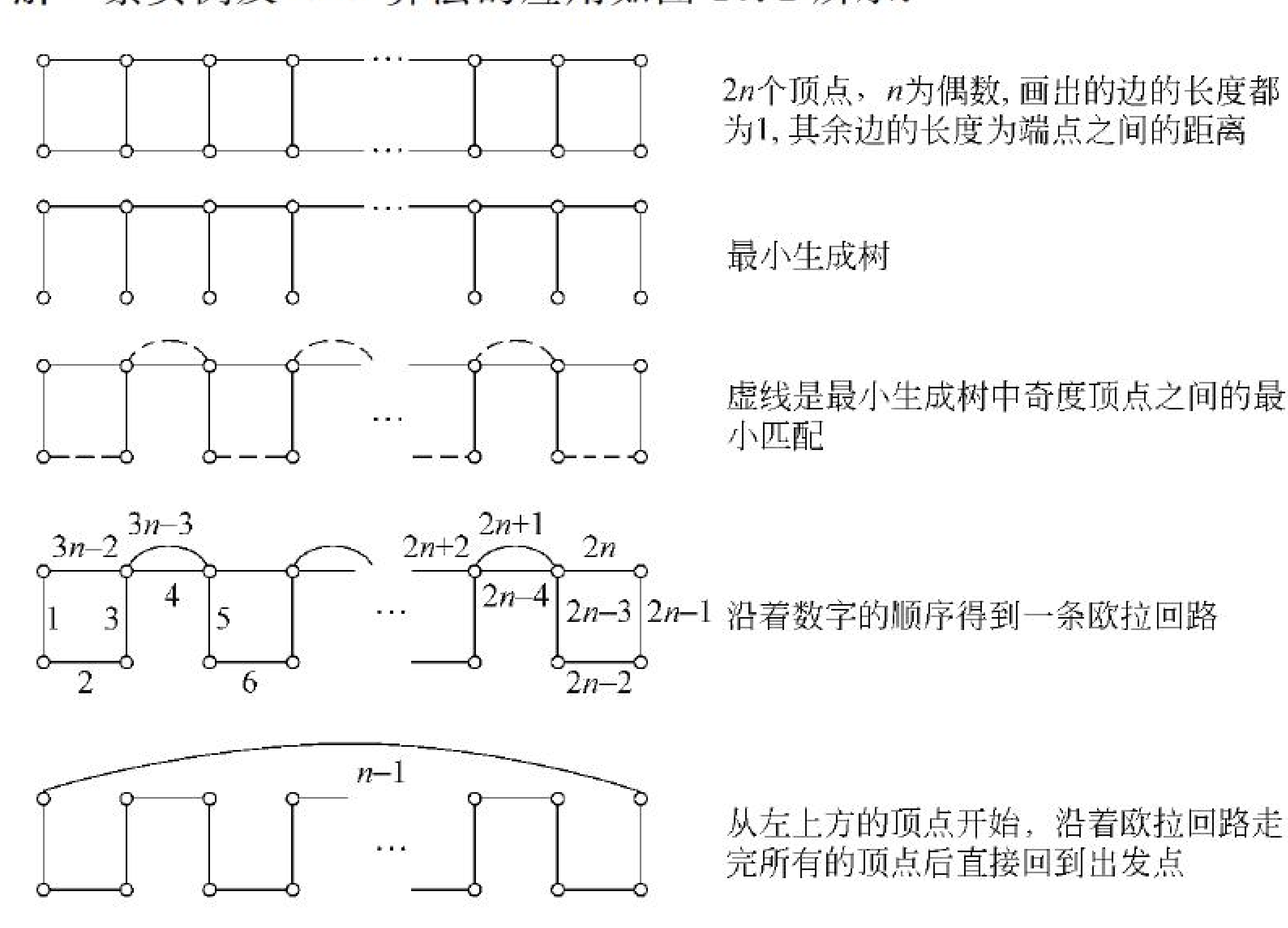


图10.2 MM算法的一个紧实例

$$\mathrm{MM}(I)=3n-2,\quad \mathrm{OPT}(I)=2n,\quad \frac{\mathrm{MM}(I)}{\mathrm{OPT}(I)}=\frac{3}{2}-\frac{1}{n}$$

10.3 证 当 $\mathrm{FF}(I)=1$ 时,显然 $\mathrm{FF}(I)=\mathrm{OPT}(I)$. 下面设 $FF(I)>1$. 记 $W=\sum_{i=1}^{n}w_i$. 因为任何两只箱子的重量之和大于 B,因此,当 $\mathrm{FF}(I)$ 为偶数时,$W>\frac{B}{2}\mathrm{FF}(I)$;当 $\mathrm{FF}(I)$ 为奇数时,设最重的箱子的重量为 B_1,则有 $W>\frac{B}{2}(\mathrm{FF}(I)-1)+B_1>\frac{B}{2}\mathrm{FF}(I)$. 故总有 $W>\frac{B}{2}\mathrm{FF}(I)$,即 $\mathrm{FF}(I)<\frac{2W}{B}$. 又显然 $\mathrm{OPT}(I)\geqslant\frac{W}{B}$,得证 $\mathrm{FF}(I)<2\mathrm{OPT}(I)$.

10.4 证 假设A是装箱问题的多项式时间近似算法,其近似比 $r<\frac{3}{2}$,要证**双机调度** $\in$ **P**,从而推出 **P**$=$**NP**.

任给**双机调度**的实例 I：n 个作业的加工时间 $t_1,t_2,\cdots,t_n$，截止时间 D. 构造对应的装箱问题的实例 I'：n 件物品的重量 $t_1,t_2,\cdots,t_n$，箱子的最大重量 D，显然，I 是是实例⇔$\mathrm{OPT}(I')\leqslant 2$. 又当 $\mathrm{OPT}(I')\leqslant 2$ 时，$\mathrm{A}(I')<\dfrac{3}{2}\times 2=3$. 由于 $\mathrm{A}(I')$ 是整数，必有 $\mathrm{A}(I')\leqslant 2$. 当 $\mathrm{OPT}(I')>2$ 时，$\mathrm{A}(I')\geqslant\mathrm{OPT}(I')>2$. 因此，$I$ 是是实例⇔$\mathrm{A}(I')\leqslant 2$.

根据上述性质，如下构造**双机调度**的算法 B：对任给的**双机调度**实例 I，n 个作业的加工时间 $t_1,t_2,\cdots,t_n$ 和截止时间为 D，首先构造对应的装箱问题的实例 I'：n 件物品的重量 $t_1,t_2,\cdots,t_n$ 和箱子的最大重量 D，然后对 I' 应用算法 A. 若 $A(I')\leqslant 2$，则 B 输出“yes”；否则 B 输出“no”. 由于 A 是多项式时间的，所以 B 也是多项式时间的.

10.5　证　根据算法，每一个顶点关联的割边数大于等于关联的非割边数. 对所有的顶点求和，每条边出现 2 次，故所有的割边数大于等于所有非割边数. 从而 $\mathrm{MCUT}(I)\geqslant\dfrac{1}{2}|E|$. 又显然，$\mathrm{OPT}(I)\leqslant|E|$. 得证 $\mathrm{OPT}(I)\leqslant 2\mathrm{MCUT}(I)$.

说明：最小割集问题是多项式时间可解的，见 7.1 节，而最大割集问题是 NP 难的.

10.6　解　递推公式

$$B(0)=\{0\}$$

$$B(i)=B(i-1)\cup\{t\mid t-t_i\in B(i-1),t_i\leqslant t\leqslant D\}\quad i=1,2,\cdots,n$$

显然，

$$\mathrm{OPT}(I)=\sum_{i=1}^{n}t_i-\max B(n)$$

算法 DP

输入：n 个作业的处理时间 $t_1,t_2,\cdots,t_n$

1. 令 $D=\left\lfloor\dfrac{1}{2}\sum_{i=1}^{n}t_i\right\rfloor$, $B(0)=0$
2. 对 $i=1,2,\cdots,n$
3. 　令 $B(i)=B(i-1)$
4. 　对 $t=t_i,t_i+1,\cdots,D$
5. 　　如果 $t-t_i\in B(i-1)$，则 $B(i)=B(i)\cup\{t\}$
6. 令 $t=\max B(n)$，$J=\varnothing$，$i=n$
7. 对 $i=n,n-1,\cdots,1$
8. 　如果 $t-t_i\in B(i-1)$，则$(J=J\cup\{i\},t=t-t_i)$
9. 　如果 $t\leqslant 0$ 则转 10
10. 输出 J

DP 的时间复杂度为 $O(nD)=O(n^2t_{\max})$，其中 $t_{\max}=\max\{t_i|i=1,2,\cdots,n\}$. 这是伪多项式时间算法.

算法 FPTAS

输入：$\varepsilon>0$ 和 n 个作业的处理时间 $t_1,t_2,\cdots,t_n$

1. 令 $t_{\max}=\max\{t_i|i=1,2,\cdots,n\}$
2. 令 $b=\max\left\{\left\lfloor t_{\max}\Big/\left(1+\dfrac{1}{\varepsilon}\right)n\right\rfloor,1\right\}$

3. 令 $t'_i = t_i / b, i = 1, 2, \cdots, n$

4. 以 $t'_i (i = 1, 2, \cdots, n)$ 为输入应用 DP

5. 输出 J

FPTAS 的时间复杂度为 $O(n^2 t'_{\max}) = O(n^2 t_{\max}/b) = O\left(n^3\left(1+\frac{1}{\varepsilon}\right)\right)$.

当 $b=1$ 时,FPTAS 得到最优解. 不妨设 $b>1, b(t'_i - 1) < t_i \leqslant bt'_i$. 对任意的 $S \subseteq \{1, 2, \cdots, n\}$,

$$b\sum_{i\in S} t'_i - b\mid S\mid < \sum_{i\in S} t_i \leqslant b\sum_{i\in S} t'_i$$

$$0 \leqslant b\sum_{i\in S} t'_i - \sum_{i\in S} t_i < b\mid S\mid \leqslant bn \qquad (*)$$

记最优解 J^*,FPTAS 得到的近似解为 J 和 $J' = \{1, 2, \cdots, n\} - J$, $\text{OPT}(I) = \sum_{i\in J^*} t_i$, $\text{FPTAS}(I) = \sum_{i\in J'} t_i$. 于是

$$\begin{aligned}\text{FPTAS}(I) - \text{OPT}(I) &= \sum_{i\in J'} t_i - \sum_{i\in J^*} t_i \\ &= \left(\sum_{i\in J'} t_i - b\sum_{i\in J'} t'_i\right) + \left(b\sum_{i\in J'} t'_i - b\sum_{i\in J^*} t'_i\right) + \left(b\sum_{i\in J^*} t'_i - \sum_{i\in J^*} t_i\right)\end{aligned}$$

由(*),第一项小于等于 0. J' 是关于 $\{t'_i\}$ 的最优解,第二项也小于等于 0. 得到

$$\begin{aligned}\text{FPTAS}(I) - \text{OPT}(I) &\leqslant b\sum_{i\in J^*} t'_i - \sum_{i\in J^*} t_i < bn \leqslant t_{\max}\Big/\left(1+\frac{1}{\varepsilon}\right) \\ &\leqslant \text{FPTAS}(I)\Big/\left(1+\frac{1}{\varepsilon}\right)\end{aligned}$$

化简得到 $\text{FPTAS}(I) \leqslant (1+\varepsilon)\text{OPT}(I)$. 得证 FPTAS 是双机调度问题的完全多项式时间近似方案.

第 11 章 随机算法

11.1 内容提要

1. 基本概念

随机变量 从概率空间到实数的映射.

分布 随机变量取各种值的概率就是分布.

期望 随机变量的加权平均值称为期望.

拉斯维加斯型随机算法 总是给出正确的结果,区别只在于运行时间的长短的随机算法.

蒙特卡洛型随机算法 有时会给出错误的答案的随机算法,蒙特卡洛型算法又可分为单侧错误的和双侧错误的两类.

弃真型错误 在求解一个判定问题时把本应接受的输入误判为拒绝.

取伪型错误 就是把本应拒绝的输入误判为接受.

ZPP 类 在计算复杂性理论中,有效的拉斯维加斯型算法也被称为 **ZPP 类**(Zero-error Probabilistic Polynomial time)算法,即零错误概率多项式时间随机算法.

BPP 类 有效的蒙特卡洛型算法也被称为 **BPP 类**(Bounded-error Probabilistic Polynomial time)算法,即错误概率有界的多项式时间随机算法,其中有效的弃真型单侧错误随机算法又称为 **RP** 类算法,有效的取伪型单侧错误随机算法则称为 **coRP** 类算法.

2. 某些重要的结果

随机变量的线性性质

$$\mathrm{E}[aX+bY]=a\mathrm{E}[X]+b\mathrm{E}[Y]$$

马尔科夫不等式 若随机变量 X 的取值非负,则对于任意 $k>0$,有

$$\Pr[X \geqslant k\mathrm{E}[X]] \leqslant 1/k$$

詹森不等式

$$\mathrm{E}[X^2] \geqslant (\mathrm{E}[X])^2$$

契比雪夫不等式 若随机变量 X 的方差为 σ,则对于任意 $k>0$,有

$$\Pr[\mid X-\mathrm{E}[X]\mid > k\sigma] \leqslant 1/k^2$$

并的界

$$\Pr[\cup_{i=1}^{n} A_i] \leqslant \sum_{i=1}^{n} \Pr[A_i]$$

切诺夫界 如果随机变量 $X_1, X_2, \cdots, X_n$ 是完全独立的，并且每个 X_i 取值在 $\{0,1\}$ 中，设 $\mu = \sum_{i=1}^{n} \mathrm{E}[X_i]$，则对于任何 $\delta > 0$，有

$$\Pr\left[\sum_{i=1}^{n} X_i \geqslant (1+\delta)\mu\right] \leqslant \left[\frac{\mathrm{e}^{\delta}}{(1+\delta)^{(1+\delta)}}\right]^{\mu}$$

和

$$\Pr\left[\sum\nolimits_{i=1}^{n} X_i \leqslant (1-\delta)\mu\right] \leqslant \left[\frac{\mathrm{e}^{-\delta}}{(1-\delta)^{(1-\delta)}}\right]^{\mu}$$

以及对任何 $c > 0$，有

$$\Pr\left[\left|\sum_{i=1}^{n} X_i - \mu\right| \geqslant c\mu\right] \leqslant 2 \cdot \exp(-\min\{c^2/4, c/2\}\mu)$$

其中 $\exp(x) = \mathrm{e}^x$.

3. 算法

快速排序.

素数检验.

多项式恒等检验.

2SAT 随机游动算法.

11.2 习　　题

11.1 证明切诺夫界.

11.2 证明在随机算法定义中，可以把平均运行时间为多项式的要求改为最坏运行时间为多项式的要求.

11.3 证明可以通过独立重复运行，把有效随机算法的错误概率降低到输入规模的指数的倒数.

11.4 给出中位数问题的随机算法，并分析期望运行时间.

11.5 利用多项式恒等检验，设计并分析一个求解二部图完美匹配问题的随机算法.

11.6 用随机游动算法求解三元可满足性问题，并且分析算法的期望运行时间.

11.3 习题解答与分析

11.1 设随机变量 $X = \sum_{i=1}^{n} X_i$，$p_i = \mathrm{E}[X_i]$，$t > 0$，则 $\mu = \mathrm{E}[X] = \sum_{i=1}^{n} p_i$.

根据**马尔科夫不等式**

$$\Pr[X \geqslant (1+\delta)\mu] = \Pr[\mathrm{e}^{tX} \geqslant \mathrm{e}^{(1+\delta)t\mu}] \leqslant \frac{\mathrm{E}[\mathrm{e}^{tX}]}{\mathrm{e}^{(1+\delta)t\mu}} \tag{11.1}$$

接下来计算 $\mathrm{E}[\mathrm{e}^{tX}]$ 的界

$$\begin{aligned}\mathrm{E}[\mathrm{e}^{tX}] &= \mathrm{E}[\mathrm{e}^{t\sum X_i}] = \mathrm{E}\left[\prod_{i=1}^{n} \mathrm{e}^{tX_i}\right] = \prod_{i=1}^{n} \mathrm{E}[\mathrm{e}^{tX_i}] \quad \text{（根据独立性）} \\ &= \prod_{i=1}^{n} (p_i \mathrm{e}^t + (1-p_i) \cdot 1)\end{aligned}$$

$$= \prod_{i=1}^{n}(1+p_i(e^t-1))$$

利用不等式：$\forall x\in R, 1+x\leqslant e^x$，有

$$E[e^{tX}] \leqslant \prod_i e^{p_i(e^t-1)} = e^{\sum_i p_i(e^t-1)} = e^{(e^t-1)\mu}$$

将 $E[e^{tx}]\leqslant e^{(e^t-1)\mu}$ 代入式(11.1)，可以得到对 $\forall t>0$，

$$\Pr[X \geqslant (1+\delta)\mu] \leqslant \frac{e^{(e^t-1)\mu}}{e^{(1+\delta)t\mu}}$$

经过简单的计算，可以知道当 $t=\ln(1+\delta)$ 时，可以使右式最小，从而使这个界更紧，代入上式，得到

$$\Pr[X \geqslant (1+\delta)\mu] \leqslant e^{(e^{\ln(1+\delta)}-1)\mu-(1+\delta)\ln(1+\delta)\mu} = (e^{\delta-(1+\delta)\ln(1+\delta)})^{\mu}$$

第一部分证毕.

类似地，

$$\Pr[X \leqslant (1-\delta)\mu] = \Pr[e^{-tX} \geqslant e^{-(1-\delta)t\mu}] \leqslant \frac{E[e^{-tX}]}{e^{-(1-\delta)t\mu}}$$

$$E[e^{-tX}] = \prod_{i=1}^{n}(1+p_i(e^{-t}-1)) \leqslant \prod_i e^{p_i(e^{-t}-1)} = e^{(e^{-t}-1)\mu}$$

$$\Pr[X \leqslant (1-\delta)\mu] \leqslant \frac{e^{(e^{-t}-1)\mu}}{e^{-(1-\delta)t\mu}}$$

经过类似的计算，可以知道当 $t=-\ln(1-\delta)$ 时，可以使右式最小，代入上式得

$$\Pr[X \leqslant (1-\delta)\mu] \leqslant e^{(e^{\ln(1-\delta)}-1)\mu-(1-\delta)\ln(1-\delta)\mu} = (e^{-\delta-(1-\delta)\ln(1-\delta)})^{\mu}$$

第二部分证毕.

为了证明第三部分，需要一个不等式

$$\forall \delta > 0, \quad \ln(1+\delta) > \frac{2\delta}{2+\delta}$$

那么，有

$$\delta-(1+\delta)\ln(1+\delta) \leqslant \frac{-\delta^2}{2+\delta}$$

$\frac{-\delta^2}{2+\delta}>-\frac{\delta}{2}$ 和 $\frac{-\delta^2}{2+\delta}>-\frac{\delta^2}{4}$ 不能同时成立，所以 $\delta-(1+\delta)\ln(1+\delta)\leqslant\max\left\{-\frac{\delta}{2},-\frac{\delta^2}{4}\right\}$，下面证明，

$$\forall\delta \in (0,1), \quad -\delta-(1-\delta)\ln(1-\delta) \leqslant -\frac{\delta^2}{4}.$$

$$\begin{aligned}-\delta-(1-\delta)\ln(1-\delta)+\frac{\delta^2}{4} &= -\delta+(1-\delta)\left(\delta+\frac{\delta^2}{2}+\frac{\delta^3}{3}+\cdots\right)+\frac{\delta^2}{4} \\ &= \left(-1+\frac{1}{2}+\frac{1}{4}\right)\delta^2+\left(-\frac{1}{2}+\frac{1}{3}\right)\delta^3+\left(-\frac{1}{3}+\frac{1}{4}\right)\delta^4+\cdots \\ &= -\left(\frac{1}{4}\delta^2+\frac{1}{6}\delta^3+\frac{1}{12}\right)\delta^4 < 0\end{aligned}$$

所以

$$-\delta-(1-\delta)\ln(1-\delta) \leqslant \max\left\{-\frac{\delta}{2},-\frac{\delta^2}{4}\right\}$$

由并的界得

$$\Pr[|X-\mu| \geqslant c\mu] \leqslant \Pr[X-\mu \geqslant c\mu] + \Pr[X-\mu \leqslant -c\mu]$$

这里 $c=1+\delta$,综合以上两部分,原命题第三部分得证.

11.2 先证明当最坏时间为多项式时间时,平均运行时间也是多项式的. 设运行时间为随机变量 T,T 的值域为 D,最坏运行时间为 T_0,则

$$\mathrm{E}[T]=\sum_{t\in D}t\Pr(t)\leqslant\sum_{t\in D}T_0\Pr(t)=T_0$$

也就是说,平均运行时间不会超过最坏运行时间,即平均运行时间也是多项式的.

再证明当平均运行时间是多项式的,则最坏时间也是多项式的.设随机算法的出错概率为 1/3.由马尔可夫不等式,$\Pr[T>10\mathrm{E}[T]]<1/10$.我们可以修改随机算法如下:每当运行时间达到或超过 $10\mathrm{E}[T]$,则终止运行,输出任意答案.这时算法的最坏运行时间是 $10\mathrm{E}[T]$,还是多项式的,而出错概率不超过 $1/3+1/10=13/30$.我们采用多执行几遍取多数次结果的办法,就可以把错误概率降到 1/3 以下.

11.3 通过独立重复运行,取多数结果为最终结果(这里我们认为结果只有两种,即正确或错误),我们知道一次独立运行,错误的概率至多为 1/3. 假设运行次数为 n,运行 n 次而得到错误结果意味着至少 $n/2$ 次运行出错. 设 0-1 随机变量 $\boldsymbol{X}_i$ 为 1 时代表第 i 次出错,为 0 时代表第 i 次运行正确. 则

$$\Pr\left[\sum_{i=1}^{n}X_i\geqslant\frac{n}{2}\right]=\Pr\left[\sum_{i=1}^{n}X_i\geqslant\left(1+\frac{1}{2}\right)\frac{n}{3}\right]$$

由切诺夫界,右式的上界为 $\mathrm{e}^{(1/2-(1+1/2)\ln(1+1/2))n/3}\approx\mathrm{e}^{-0.036n}$,故原命题成立.

11.4 我们将这个问题一般化,定义如下的选择问题:

输入:一个包含 n 个(不同的)数的集合 A 和一个数 i,$1\leqslant i\leqslant n$.

输出:元素 $x\in A$,它恰大于 A 中的其他的 $i-1$ 个元素.

显然,中位数问题也属于选择问题. 这里运用快速排序的思路. 与快排不同的是,快排需要对枢轴元素的两边进行操作,而选择问题只需要对其中的一边进行操作.

算法步骤:

1. 若数组包含 0 或 1 个元素,则返回.
2. 从数组中随机选择一个元素作为枢轴元素.
3. 把数组元素分为两个子数组,并且按照 B、枢轴元素、C 的顺序排列:

   B:包含比枢轴元素小的元素.

   C:包含比枢轴元素大的元素.
4. 设枢轴元素位于第 k 位,

   若 k 等于 i,则返回 $A[i]$;

   若 k 大于 i,则在 B 中递归地找第 i 小元素;

   若 k 小于 i,则在 C 中递归地找第 $i-k$ 小的元素.

对该算法的分析如下:设对 n 个数字组成的数组,找第 i 大的元素,需要的时间是 $T(n,i)$,设其最大的为 $T(n)$,这是一个随机变量,下面得到 $T(n)$ 期望的上界.

由于在算法步骤 2 中枢轴元素是随机选取的,所以对每一个 $k(0\leqslant k\leqslant n-1)$,子数组 B 中有 k 个元素的概率为 $1/n$,定义指示器随机变量 X_k 为

$$X_k=\begin{cases}1 & \text{子数组 } B \text{ 中恰好有 } k \text{ 个元素}\\0 & \text{否则}\end{cases}$$

假定元素的值不同,因此有

$$E[X_k]=1/n$$

为了得到一个上界，假定第 i 个元素总是被划分到较大的一边．当 $X_k=1$ 时，可能要递归处理的子数组的大小为 k 和 $n-k-1$，因此得到递归式：

$$T(n)\leqslant O(n)+\sum_{k=0}^{n-1}X_k\cdot T(\max(k,n-k-1))$$

取期望值，得到

$$\begin{aligned}E[T(n)]&\leqslant O(n)+E\Big[\sum_{k=0}^{n-1}X_k\cdot T(\max(k,n-k-1))\Big]\\&=O(n)+\sum_{k=0}^{n-1}E[X_k\cdot T(\max(k,n-k-1))]\quad\text{（期望的线性性）}\\&=O(n)+\sum_{k=0}^{n-1}E[X_k]\cdot E[T(\max(k,n-k-1))]\quad\text{（期望的独立性）}\\&=O(n)+\sum_{k=0}^{n-1}\frac{1}{n}\cdot E[T(\max(k,n-k-1))]\\&=O(n)+\frac{2}{n}\sum_{k=n/2}^{n-1}E[T(k)]\end{aligned}$$

由数学归纳法，不难证明 $E(T(n))=O(n)$．即在平均情况下，我们能在线性时间内计算出选择问题．特别地，可以找出不同数字组成数组的中位数．

11.5 假设二部图 $G=\langle U,V,E\rangle$，U 和 V 是两个自身不相交的集合，即 U 和 V 都是独立集，E 是连接 U 和 V 中的顶点的边集．设 $|U|=|V|=n$，$U=\{u_1,u_2,\cdots,u_n\}$，$V=\{v_1,v_2,\cdots,v_n\}$．定义一个 $n\times n$ 的矩阵 $\boldsymbol{M}$，定义

$$M_{i,j}=\begin{cases}0 & \text{若}(u_i,v_j)\notin E\\ x_{i,j} & \text{若}(u_i,v_j)\in E\end{cases}$$

令 $\mathrm{Det}(\boldsymbol{M})$表示 $\boldsymbol{M}$ 的行列式，则我们有如下的命题：

命题 11.1 $\mathrm{Det}(\boldsymbol{M})\equiv 0\Leftrightarrow G$ 中不存在完美匹配．

证 对行列式，我们有如下的展开式，

$$\mathrm{Det}(\boldsymbol{M})=\sum_{\pi\in S_n}(-1)^{\mathrm{sgn}(\pi)}\prod_{i=1}^{n}M_{i,\pi(i)}$$

其中 S_n 是所有在集合$\{1,2,\cdots,n\}$上的置换的集合，$\mathrm{sgn}(\pi)$是置换 π 的符号（当 π 是偶置换时为正，当 π 是奇置换时为负；而置换的奇偶性等于把该置换通过对换变为恒等排列所需对换次数的奇偶性）．注意到所有 G 中的完美匹配都与集合上的置换存在着一一对应，即 π 与 G 中匹配$\{(u_1,u_{\pi(1)}),(u_2,u_{\pi(2)}),\cdots,(u_n,u_{\pi(n)})\}$构成一一对应．注意，如果该完美匹配不存在，那么相应 π 的行列式项为 0，即

$$\mathrm{Det}(\boldsymbol{M})=\sum_{m_\pi\in P}(-1)^{\mathrm{sgn}(\pi)}\prod_{i=1}^{n}M_{i,\pi(i)}$$

P 是所有完美匹配的集合，m_π 是 π 所对应的匹配．当 P 为空集时，上式显然恒等于 0，即如果没有完美匹配，则上述行列式恒为 0．如果 G 有完美匹配，那么 π 所对应的 P 中的项不恒为 0，由于没有其他项与该项有完全相同的变量，所以存在一组赋值，使该行列式不为 0．

下面对该行列式运用多项式恒等检验的算法，注意，直接将行列式展开可能有 $n!$项，而

将变元换成数字则可以用高斯消去法在 $O(n^3)$ 内计算. 注意,行列式表示的多项式的次数至多为 n,故用多项式恒等检验解决完美匹配是否存在的算法如下:

输入:用 $n\times n$ 的矩阵 $\boldsymbol{M}$ 表示的多项式 $r(x_1,\cdots,x_{n^2})$

输出:$r(x_1,\cdots,x_{n^2})$是否恒等于 0

1. 从$\{1,\cdots,10n^2\}$中随机选择 n^2 个自然数 $a_1,\cdots,a_{n^2}$(可重复).
2. 从$\{1,\cdots,n^4\}$中随机选择自然数 k.
3. 在模 k 算术下,计算 $y=\mathrm{Det}(\boldsymbol{M})(\bmod k)=r(a_1,\cdots, a_{n^2})(\bmod k)$.
4. 若 $y=0$,则输出"$r(x_1,\cdots, x_{n^2})$恒等于 0".
5. 否则,输出"$r(x_1,\cdots, x_{n^2})$不恒等于 0".

则类似多项式恒等检验的分析,可以得到当 $r(x_1,\cdots, x_{n^2})$恒等于 0 时,算法输出正确结果;当 $r(x_1,\cdots, x_{n^2})$不恒等于 0 时,算法至少以 $9/(80\ln n)$的概率输出正确结果.

由于判定问题可以解决,所以利用搜索归约为判定,一旦可以找到一组赋值使行列式不为 0,则可以找到对应的完美匹配.

11.6 求解三元布尔可满足性问题的随机游动算法.

输入:一个有 n 个变元和 m 个子句的合取范式,每个子句至多有 3 个文字

输出:一个可满足赋值,或者宣布没有可满足赋值

1. 任意给所有变量赋值.
2. 如果当前赋值是可满足赋值,则输出这个赋值,算法结束.
3. 均匀随机选一个不满足子句,从中均匀随机选一个文字,改变该文字的赋值.
4. 重复第 2 步和第 3 步 $m2^{n+3}$ 次,若一直没有找到可满足赋值,则宣布没有可满足赋值.

下面证明如下命题:当输入公式是不可满足的,算法无法找到可满足赋值,因此将宣布不可满足. 当输入公式可满足时,算法有可能找不到可满足赋值而出错. 我们证明算法出错的概率为 $1/2^m$.

假设输入公式可满足,设 a^* 是某个可满足赋值,a_t 是算法在执行 t 遍第 3 步以后的赋值,X_t 等于 a^* 和 a_t 赋值相同的变量个数. 令变量数为 n,则当 $X_t=n$ 时 $a_t=a^*$,算法将找到可满足赋值 a^* 而停止. 注意在 $X_t=n$ 之前,算法也可能找到别的可满足赋值而停止.

当 $X_t=0$ 时,a^* 和 a_t 赋值相同的变量个数为 0. 修改任何一个变量的赋值,都会让 a^* 和 a_t 赋值相同的变量个数增加 1 个,即 $X_{t+1}=1$,所以 $\Pr[X_{t+1}=1\mid X_t=0]=1$. 当 $X_t=j$ 且 $0<j\leqslant n-1$ 时,a^* 和 a_t 赋值相同的变量个数为 j. 修改某一个变量的赋值,只会让 a^* 和 a_t 赋值相同的变量个数增加 1 个或减少 1 个,即 $X_{t+1}=j+1$ 或 $X_{t+1}=j-1$. 假设算法选择修改子句 $C=a\vee b\vee c$ 的一个变量的赋值,根据算法的定义可以知道,在赋值 a_t 之下 C 是不满足子句,所以在赋值 a_t 之下 $a=b=c=\mathrm{False}$. 由于假定 a^* 是可满足赋值,因此在a^* 下,a、b、c 有另外的 7 种赋值情况. 因此随机选择改变 a 或 b 的赋值时,至少有 1/3 机会让 a^* 和 a_t 赋值相同的变量个数增加 1 个. 所以 $\Pr[X_{t+1}=j+1\mid X_t=j]\geqslant 1/3$,$\Pr[X_{t+1}=j-1\mid X_t=j]\leqslant 2/3$. 注意$\{X_0,X_1,X_2,\cdots\}$可能不是马氏链,因为 $\Pr[X_{t+1}=j+1\mid X_t=j]$可能不是常数,这个值可能是 1/3、2/3 或 1,这与 a^* 和 a_t 赋值相同的变量具体是哪些变量有关.

为了利用马氏链来进行分析,我们定义一个真正的马氏链$\{Y_0,Y_1,Y_2,\cdots\}$,如下:

$$Y_0 = X_0$$
$$\Pr[Y_{t+1} = 1 \mid Y_t = 0] = 1$$
$$\Pr[Y_{t+1} = j+1 \mid Y_t = j] = 1/3$$
$$\Pr[Y_{t+1} = j-1 \mid Y_t = j] = 2/3$$

注意

$$\Pr[Y_{t+1} = j+1 \mid Y_t = j\ ] = 1/3 \leqslant \Pr[X_{t+1} = j+1 \mid X_t = j]$$
$$\Pr[Y_{t+1} = j-1 \mid Y_t = j\ ] = 2/3 \geqslant \Pr[\ X_{t+1} = j-1 \mid X_t = j\ ]$$

从任意状态出发,Y_t 将比 X_t 花费更长的期望时间才能进入状态 n. 下面分析 Y_t 进入状态 n 所需要的期望时间,以此作为算法期望运行时间的上界. 注意马氏链 $Y=\{Y_0,Y_1,Y_2,\cdots\}$ 可以用如图 11.1 所示的带权有向图表示.

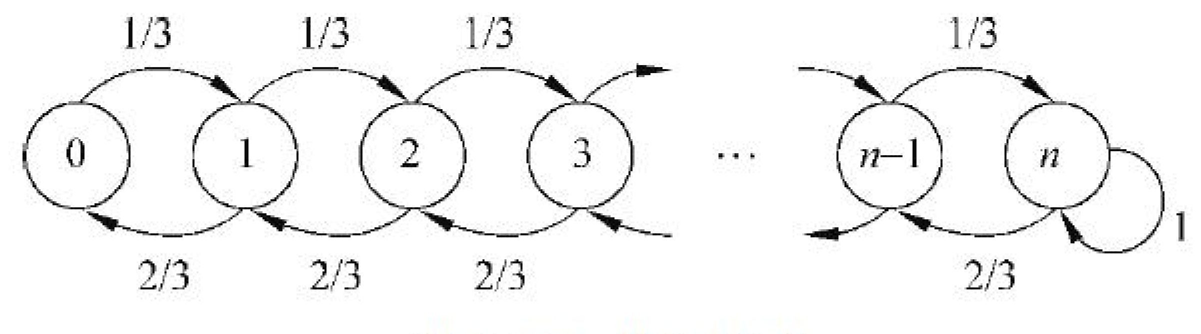


图 11.1　马氏链 Y

设 h_j 表示从状态 j 到达状态 n 的期望运行时间,则有 $h_n=0$ 和 $h_0=h_1+1$. 对于其他 j 值,有

$$h_j - 1 = \frac{1}{3}h_{j+1} + \frac{2}{3}h_{j-1}$$

将上式变形,得到

$$h_{j+1} - h_j - 3 = 2(h_j - h_{j-1} - 3)$$

所以 $h_{j+1}-h_j-3$ 为等比级数,将 $h_1-h_0-3=-4$ 代入上式,得 $h_{j+1}-h_j-3=-2^{j+2}$,两边同时对 j 求和,得到

$$h_0 = 2^{n+2} - 3n - 4$$

由马尔科夫不等式,就有

$$\Pr[X \text{ 从 } X_0 \text{ 到达 } n \text{ 的时间} > 2^{n+3}] \leqslant 1/2$$

即若以 2^{n+3} 步为一个阶段,则在一个阶段中算法找不到可满足赋值的出错概率不超过 1/2. 当重复执行每个阶段 m 次,即总共执行 $m2^{n+3}$ 步时,算法仍然找不到可满足赋值的出错概率就小于 $1/2^m$.

第 12 章 处理难解问题的策略

12.1 内容提要

1. 基本概念

二元可满足性(2SAT) 任给一个二元合取范式 F,问 F 是否可满足.

霍恩(Horn)子句 每个子句中正文字(不带否定号的变量)至多出现一次.

霍恩公式 由霍恩子句构成的公式.

霍恩 SAT 问题 输入限制为霍恩公式的 SAT 问题.

固定参数算法 输入中带有一个参数 k,当输入规模为 n 时运行时间为 $O(f(k)n^c)$的算法.

指数时间假设 kSAT 的指数时间算法的底数不能任意地小.

随机化策略 随机选择初始解.

重启策略 算法在不同的初始解上重新开始多运行几次.

后门变量 存在少数一些变量,称为**后门变量**(backdoors),只要确定了这些变量的取值,整个问题就非常容易解决.

相变现象 存在参数的一个临界值,实例的性质在临界点两侧发生了很大的突变,具有某个性质的概率在一侧随实例规模趋于无穷而渐近为 0,在另一侧则随实例规模趋于无穷而渐近为 1.

RB 模型 对 B 模型的修订,是首个在理论上严格地证明了存在既有精确相变现象又有难解实例的 NP 完全问题的随机模型.

量子算法 基于量子物理学原理在量子计算机上运行的算法.

2. 某些重要的结果

2SATP.

霍恩公式可满足性问题是易解的.

哈密顿回路问题在 $G(n,1/2)$上有有效算法.

3. 算法

2SAT 算法.

求解 HornSAT 的多项式时间算法.

顶点覆盖问题的固定参数算法.

3SAT 的指数时间改进算法.

模拟退火法.

哈密顿回路问题在 $G(n,1/2)$ 上的有效算法.
RB 模型随机实例产生算法.
产生隐藏着解或最优解的算例的算法.
用 RB 模型产生最大团难解算例.
随机 3SAT 的 SP 算法.
道奇算法.

12.2 习　　题

12.1 证明最小顶点覆盖、最大团、最大独立集在树上都是易解的.

12.2 所谓区间图是这样的图,图的每个顶点都对应于某条固定直线上的一段区间,两个顶点相邻当且仅当其对应的区间相交(重叠). 证明区间图的顶点着色是易解的.

12.3 所谓圆弧图是这样的图,图的每个顶点都对应于某条固定圆周上的一段圆弧,两个顶点相邻当且仅当其对应的圆弧相交(重叠). 研究圆弧图的顶点着色问题,它是 NP 完全的,还是易解的?

12.4 证明圆弧图的顶点着色有固定参数算法,其中参数 k 为着色的色数.

12.5 给出比主教材正文中结果更好的 3SAT 的改进的指数时间确定算法.

12.6 利用随机游动算法给出 kSAT 的改进的指数时间随机算法.

12.7 给出 kSAT 的改进的指数时间确定算法.

12.8 针对某个实际问题应用一下模拟退火策略.

12.9 完成教材中定理 12.4 证明中的细节,并证明算法实际可在 $O(n^2)$时间内运行.

12.10 按照最大团难解实例的产生方法,自己产生一个难解实例,并试验求解. 你能求出的最大团有多少个顶点?

12.11 产生随机 3SAT 的一些实例,用 SP 算法求解这些实例,观察 m/n 最大到多少时 SP 算法仍然有效.

12.12 验证教材 12.8 节中介绍的量子门都是酉变换.

12.13 把道奇算法推广到函数 $f:\{0,1\}^n\{0,1\}$上,即假设函数 f 要么在所有输入上都取相同的值,要么在一半的输入上取值 0,在另一半的输入上取值 1,设计量子算法区分这两种情况.

12.3 习题解答与分析

12.1 树上的最小顶点覆盖算法:要计算树上的最小顶点覆盖,只需要贪心策略即可. 对于点覆盖,按照定义,对图中的任意一条边,必须至少有一个相关的顶点在点覆盖中. 由于对于叶子结点,每个顶点只能覆盖一条边,所以第一步先把所有叶子结点去掉而将和叶子结点直接相邻的顶点加入点覆盖,然后将这些顶点已覆盖的边去掉. 留下来的图必然还是一棵树. 继续上面的操作,直到图为空图. 由于每次剩下树的叶子结点至少有一个,所以外层循环至多有 n 次,每次最多遍历一遍树,复杂性为 $O(n)$,故算法复杂性为 $O(n^2)$,即是易

解的.

关于最大团,按照定义,只要树有边,则为任意一个 K_2;否则最大团为孤立的顶点. 显然是易解的.

树上的最大独立集算法:按照定义不难发现,对于任意的一个图,其点覆盖的余集便为独立集,独立集的余集便为点覆盖,故最大独立集对应最小点覆盖. 所以只需要先算出最小顶点覆盖,然后求余集即可. 所以也是易解的.

12.2 首先,题目中所说的区间图是一种弦图,即任何长度大于等于 4 的圈都有一条弦. 如在图中发现了一个圈 $abcda$,则必须有边 ac 或有边 bd. 这个用区间图的性质易证. 弦图有一个很好的性质,即可以找到一个顶点序列,使得任一顶点 v 的邻域与 v 后边顶点的交集是一个团. 可以按照该序列进行着色,可以证明区间图的色数为该图最大团的顶点数. 下面给出一些细节的证明.

命题 12.1 区间图是弦图.

证 用反证法. 假设存在一条大于等于 4 的圈 $abcd\cdots a$,该圈上没有弦,即该圈中的点除了与两边的点相连外不存在其他的边(例如边 ac),那么按照区间图的定义,相邻的两个区间相交非空,但不相邻的两个区间交集为空集,显然矛盾.

命题 12.2 弦图的导出子图也是弦图.

证 按照导出子图的定义,只要该导出子图包含任意的一个圈,必然包含两端都在该圈上的点的边. 由于原图是弦图,任意长度大于等于 4 的圈都会有一条弦,所以导出子图的任意长度大于等于 4 的圈也会有这条弦,故导出子图为弦图.

命题 12.3 一个图是弦图的充分必要条件是任意的极小点割集是一个团.

证 必要性. 令 $G=<V,E>$ 是一个弦图,令 S 是 G 的一个极小点割集. 设 x 和 y 是 S 中的两个点,下面证明 xy 之间有一条边. 设 A、B 为 $G[V-S]$ 的不同的连通分支,由 S 的极小性,仅通过 A 中的点有一条从 x 到 y 的路径. 令 p_1 为这些路径中最短的一条. 同理,令 p_2 为仅通过 B 中的点,且通过 x 到 y 的路径中的最短的一条,则连接 p_1 和 p_2 便是一个长度不小于 4 的圈. 又因为没有边直接联系 A、B 间的点,所以 x 和 y 之间必须有一条边. 由 x 和 y 的任意性,可知 S 中的任意两个点都有边相连,故 S 是一个团.

充分性. 设 G 中的任意极小点割集都是一个团. 假设 G 不是弦图. 设 $w,x,y,z_1,z_2,\cdots,z_k,w$ 是 G 中的无弦且长度大于等于 4 的圈($k\geqslant 1$),则任意将 w 和 y 分开的点割集必须含有 x 和至少一个 z_i,$1\leqslant i\leqslant k$. 由于所有的极小点割集都是团,所以 x 和 z_i 有边相连,与该圈无弦矛盾.

命题 12.4 如果图 G 的一个顶点 v 的邻域是一个团,称 v 是单纯的. 那么一个弦图 G 或者是团,或者至少有两个不相邻的单纯的顶点.

证 假设 G 不是团,我们对 G 的顶点数进行归纳. 假设 $n>2$,并且对于所有的少于 n 个顶点的图 G,结论均成立. 令 a 和 b 是图 G 中的两个不相邻的顶点,S 是将 a 和 b 分开的极小点割集,则导出子图 $G[V-S]$ 至少有两个不同的连通分支. 设 $G[A]$ 和 $G[B]$ 分别为包含 a 和 b 的连通分支. 注意到 $G[A\cap S]$ 是一个顶点数小于 n 的弦图,则其或者是团,即 A 中有一个顶点是单纯的;或者有两个不相邻的单纯的顶点. 由命题 12.3,S 是一个团,所以至少有一个顶点在 A 中. 同理,至少有一个单纯的顶点在 B 中,所以 G 有两个不相邻的单纯的顶点.

命题 12.5 弦图 G(顶点集非空)至少有一个单纯的顶点.

证 由命题 12.4,如果 G 为团,则每一个顶点是单纯的;如果 G 不是团,则命题 12.4 说明了 G 至少有一个单纯的顶点.

由命题 12.5,可以得到区间图着色的一个多项式算法,即首先在图中找到一个单纯的顶点,将其和其邻域中的点着色,这是一个团的着色. 然后将该顶点去掉,将已用的颜色记录. 之后剩余的 $n-1$ 个顶点的图也是弦图,可以再找到一个单纯的顶点. 如果其已着色,则将其邻域着色,优先选择已经记录的颜色,并保证没有冲突;否则,将该点优先用记录的颜色着色,再对其邻域中的点着色. 显然,用该算法着色,用的颜色数为该图的最大团的顶点数. 显然,算法是多项式的.

12.3 圆弧图的顶点着色问题是 NP 完全的. 为了证明这个命题,需要一系列的准备,首先介绍一个 NP 完全问题,即**不相交路径问题**.

在不相交路径问题中,给定一个有向图 G,和一个始点-终点对的集合$\{<s_1,t_1>,<s_2,t_2>,\cdots,<s_k,t_k>\}$,要求找到 k 条边不相交的路径 $P_1,P_2,\cdots,P_k$,使得路径 P_i 连接 s_i 和 t_i. 这个问题通常描述如下:

输入:原图 G 和所求的图 H,它们是顶点集相同的有向图.

输出:对每一条 H 中的边 e,在 G 中找到一条路径 P_e,使得这些路径互相是边不相交的,并且 P_e 和 e 合起来形成一个有向圈.

在这里引入一个定理(参见文献 Even S., Itai A., Shamir A. *On the complexity of timetable and multicommodity flow problems*, Proc. of FOCS 1975, 184—193):即使 G 是无圈的,不相交路径问题也是 NP 完全的.

一个有向图是欧拉图的充分必要条件是对该图的任意一个顶点,其出度等于入度. 设 $G+H$ 表示把图 G 的边和图 H 的边合并后得到的图(图 G 和图 H 有相同的顶点集). 欧拉图的不相交路径问题是指在 $G+H$ 是欧拉图的假设下的不相交路径问题. 将这个问题简称为**无圈欧拉图的不相交路径问题**.

首先证明下述命题.

命题 12.6 当 G 是无圈的且 $G+H$ 是欧拉图时,不相交路径问题仍然是 NP 完全的.

证 在无圈图 G 中,首先添加两个顶点 s、t 和一些新的边,使得 $G+H$ 是欧拉图. 如果顶点 x 在 $G+H$ 中入度 $\delta^-_{G+H}(x)$ 小于出度 $\delta^+_{G+H}(x)$,则向 G 中添加$(\delta^+_{G+H}(x)-\delta^-_{G+H}(x))$个平行边$\overrightarrow{sx}$. 如果 $\delta^-_{G+H}(x)>\delta^+_{G+H}(x)$,则向 G 中添加$(\delta^-_{G+H}(x)-\delta^+_{G+H}(x))$个平行边$\overrightarrow{xt}$. 记 G' 为 G 经修改后的图,H'为 H 加上 $\delta^+_{G'}(s)=\delta^-_{G'}(t)$条平行边$\overrightarrow{ts}$. 显然,$G'+H'$为欧拉图. 可以证明如下结论:对不相交路径问题来说,(G,H)有解当且仅当(G',H')有解. 首先,如果(G',H')有解,由于 H 中的要求不能利用 G'中的新边,故(G,H)有解. 其次,如果(G,H)有解,由于 $G'+H'$是欧拉图,将其中的某些圈删去后仍然是欧拉图. 故将(G,H)解对应的圈删去后,仍然是一个欧拉图. 此欧拉图可以分解为边不交的圈的并,由于每一个圈至多利用一个$\overrightarrow{ts}$,所以,所有的$\overrightarrow{ts}$要求,即 H'比 H 多出来的要求都可以被满足.

为了将上述的不相交路径问题归约到圆弧图的染色问题,需要如下命题.

命题 12.7 如果 $G+H$ 是欧拉图,G 是无圈图,则不相交路径问题的每一个解都用了 G 中的所有的边.

证 考虑上述不相交路径问题的一个解,将所有包含解中的圈都删掉. 由于 $G+H$ 是

欧拉图,而删去的是有向的圈,所以剩下的图也是欧拉图. 并且,由于所有的 H 中的边都被删去了,所以剩下的图是 G 的子图,并且是无圈的,而无圈的欧拉图必然没有边,这意味着所有的 G 中的边都被删去了.

下面将圆弧图是否可以 k 着色的问题归约到上述的无圈欧拉不相交路径问题.

设 $1,2,3,\cdots,n$ 是 G 中顶点的一个拓扑序列,首先构造一个圆弧图如下:令 $0,1,2,\cdots,n$ 是圆上按顺时针排列的点. 对每一条 G 中的边 $\overrightarrow{xy}$,不越过 0,即按顺时针方向构造一条弧从 x 到 y. 在这里假设所有的 $\overrightarrow{xy}\in H$,都有 $x>y$;否则一定无解. 对每一条 H 中的边 $\overrightarrow{xy}$,越过 0,即按顺时针方向构造一条弧从 x 到 y,设 k 为 H 中边的个数. 下面证明,构造的圆弧图为 k 着色的当且仅当原来的不相交路径问题有解.

首先,假设原不相交路径问题有解,则根据引理,每一条边都被一条路径用到,因此 $G+H$ 可以被严格地分成不相交的 k 个圈. 对每一个圈,用同一种颜色染对应的圆弧图,显然这种颜色对应的区间互不相交,所以用同一种颜色着色没有冲突. 因此,我们得到了对应圆弧图的一种 k 着色方案.

现在,假设对应的圆弧图为 k 可着色的. 注意到 0 恰好被 k 条弧覆盖. 由于 $G+H$ 是欧拉图,所以,任意一个圆弧区间都被 k 条弧覆盖. 相应于 H 的弧因为同时要过点 0,所以必然有不同的颜色. 对于任意的 $\overrightarrow{xy}\in H$,考虑那些和 $\overrightarrow{xy}$ 对应的弧有相同颜色的弧,它们必然覆盖了所有的区间且没有冲突,即相应的 G 中是对应 $\overrightarrow{xy}$ 的一条从 y 到 x 的路径. 所有的 k 种颜色代表了 k 条不相交的路径,它的不相交性是由一条弧只有一种颜色保证的.

显然,上述的归约是多项式的,将 NP 完全的"无圈欧拉不相交路径问题"归约到圆弧图着色问题,故圆弧图着色问题是 NP 完全的.

12.4 首先,形式化圆弧图着色问题,并介绍对称群乘积问题,并证明两者在多项式意义下是等价的.

一族圆弧 F 是一个集合 $\{A_1,A_2,\cdots,A_n\}$,每一个 A_i 是一个正整数组成的有序对 $<a_i,b_i>$,其中,$a_i\neq b_i$. 令 $m=\max_i\{a_i,b_i\}$,将圆弧等分为 m 段,顺时针标记为 $1,2,\cdots,m$,每个 A_i 可以看成在圆上按顺时针方向从点 a_i 到 b_i. 由于我们只关心圆弧是否相交,所以不妨设 $m\leqslant 2n$. 定义 A_i 的延拓

$$\mathrm{sp}(A_i)=\begin{cases}\{a_i+1,a_i+2,\cdots,b_i\} & \text{如果 } a_i<b_i\\ \{a_i+1,\cdots,m,1,2,\cdots,b_i\} & \text{如果 } a_i>b_i\end{cases}$$

如果 $\mathrm{sp}(A_i)\cap\mathrm{sp}(A_j)\neq\varnothing$,则称圆弧 A_i 和 A_j 相交. 注意,按照上述定义,如果两条弧仅仅端点相同,则它们是不相交的. 现在,圆弧图便是 $G=<F,E>$,其中 $(A_i,A_j)\in E$ 当且仅当 A_i 和 A_j 相交.

现在,形式化地定义圆弧图的着色问题,即给定一族圆弧 F 和一个正整数 K,是否可以将 F 分成 K 个子集,使得任意一个子集的任意两个圆弧不相交.

下面定义对称群的乘积(复合)问题,定义 S_K 为所有 $\{1,2,\cdots,K\}$ 上的置换组成的对称群. $X\subseteq\{1,2,\cdots,K\}$,令 S_X 为 S_K 的子群,S_X 由只变换 X 中元素的那些置换组成. 设 P_1 和 P_2 是 S_K 的子集,定义 $P_1\cdot P_2$ 为集合 $\{\pi\in S_K\mid\pi=\pi_1\cdot\pi_2,\pi_1\in P_1,\pi_2\in P_2\}$. 现在,**对称群的置换问题**(The Word Problem for Products of Symmetric Groups,WPPSG)的定义如下.

WPPSG:给定 K,一些子集 $X_1,X_2,\cdots,X_m\subseteq\{1,2,\cdots,K\}$,一个置换 $\pi\in S_K$,$P=S_{X_1}\cdot$

$S_{X_2}\cdots S_{X_m}$，判断是否 $\pi\in P$？

为了证明圆弧图着色有固定参数算法，首先证明如下的命题.

命题 12.8 WPPSG 和圆弧图着色是多项式相关的.

证 首先，假设一个 WPPSG 的实例 $K,X_1,X_2,\cdots X_m,\pi$，下面在多项式时间内构造一族圆弧 F，使得 F 可 K 着色当且仅当 $\pi\in P$.

不失一般性，假设每一个整数 $i\in\{1,2,\cdots,K\}$ 都至少在一个 S_i 出现. 若 i 不在任意一个中出现，当 $\pi(i)\neq i$ 时，必然 $\pi\notin P$；否则，删掉 i，可以获得一个更小的实例. 圆弧族 F 由 $1,2,\cdots,K+m$ 这些点组成的有序对构成. 对每个 $i\in\{1,2,\cdots,K\}$，F 包含圆弧集 F_i 和单个圆弧 C_i，其中 F_i 由包含 i 的集合 X_i 决定，C_i 由 $\pi^{-1}(i)$决定. 令 $l_i[1]$，$l_i[2]$，$\cdots$，$l_i[k(i)]$表示包含 i 的集合 X_i 的下标，以递增序排列. F_i 由 $k(i)$条弧组成：

$$
\begin{aligned}
A_{i1} &= < i,K+l_i[1] > \\
A_{i2} &= < K+l_i[1],K+l_i[2] > \\
A_{i3} &= < K+l_i[2],K+l_i[3] > \\
&\vdots \\
A_{i,k(i)} &= < K+l_i[k(i)-1],K+l_i[k(i)] >
\end{aligned}
$$

注意，这些弧的延拓是互相不交的，并且这些弧的延拓的并包含了所有从 $i+1$ 到 $K+l_i[k(i)]$ 的点. 弧 C_i 的定义为$C_i=<K+l_i[k(i)],\pi^{-1}(i)>$，令 $C=\{C_1,C_2,\cdots,C_K\}$，则圆弧族 F 的定义如下：

$$F=\bigcup_{i=1}^{k} F_i \cup C$$

显然，F 的构造是多项式的，现在证明 F 可 K 着色当且仅当 $\pi\in P$.

考虑所有对 F'的 K 着色方案. 与上边的构造不同的是，我们用点 $1,2,\cdots,K+m+1$，将每一条弧 $C_i=<K+l_i[k(i)],\pi^{-1}(i)>\in C$ 替换成两条弧，分别为$<K+l_i[k(i)],K+m+1>$和$<K+m+1,\pi^{-1}(i)>$. 令 $F_i'=F_i\cup\{<K+l_i[k(i)],K+m+1>,<K+m+1,i>\}$，而 $F'=\bigcup_i F_i'$，$1\leqslant i\leqslant K$，现在 F_i'由两两不相交的圆弧构成，且这些圆弧的延拓的并包含了所有的点 p，$1\leqslant p\leqslant K+m+1$.

任意 F'的 K 着色可以用一族函数 σ_p 表示，其中 $1\leqslant p\leqslant K+m+1$，其中 $\sigma_p(j)=i\in\{1,2,\cdots,K\}$意味着，颜色 j 被赋予了 F_i'中延拓后包含点 p 的圆弧. 因此，σ_p 是$\{1,2,\cdots,K\}$上的一个置换. 不失一般性，可以假设对所有的 j，$\sigma_1(j)=j$，也就是说，圆弧$<K+m+1,i>$被染成颜色 i. 并且，我们观察到在每一个集合 F_i'中，$<K+m+1,i>$和$<i,K+l_i[1]>$两条弧都和 $K-1$ 条弧$<K+m+1,k>$，$i+1\leqslant k\leqslant K$ 和$<k,K+l_k[1]>$，$1\leqslant k\leqslant i-1$ 相交，因为这 $K-1$ 条弧两两相交，所以，对 $i\in\{1,2,\cdots,K\}$，$<K+m+1,i>$和$<i,K+l_i[1]>$有同样的颜色，即有

$$\sigma_1=\sigma_2=\cdots=\sigma_{K+1}$$

现在研究如何从 σ_p 到σ_{p+1}，$K+1\leqslant p<K+m+1$. 如果 $\sigma_p(j)=i$ 并且在 F_i'中覆盖点 p 的弧也覆盖 $p+1$，那么 $\sigma_p(j)=\sigma_{p+1}(j)=i$. 因此，$\sigma_p(j)\neq\sigma_{p+1}(j)$唯一的可能性在于 $F'_{\sigma_{p(j)}}$ 包含一条在 p 点结束的弧. 然而，根据我们的构造，这些集合 F_i'满足 $i\in X_{p-K}$. 所以，$\sigma_{p+1}=\sigma_p\cdot\pi_{p-K}$，其中，$\pi_{p-K}\in S_{X_{p-K}}$.

因此，从 F'的 K 着色导出的最后一个置换σ_{K+m+1}可以被分解为 $\pi_1\cdot\pi_2\cdot\cdots\cdot\pi_m$，其中

$\pi_1 \in S_{X_i}$. 因此,所有的 σ_{K+m+1} 构成了集合 $P = S_{X_1} \cdot S_{X_2} \cdots S_{X_m}$.

现在来考虑对 F' 和 F 着色的不同,对每一个 C_i,被分成了两个部分,$\langle K+l_i[k(i)], K+m+1\rangle$ 在 F'_i 中,$\langle K+m+1, \pi^{-1}(i)\rangle$ 在 $F'_{\pi^{-1}(i)}$ 中,要想让这两个部分的颜色相同,必须有 $\sigma_{K+m+1}^{-1}(i) = \sigma_1^{-1}(\pi^{-1}(i))$. 由于这个等式对所有 $i \in \{1,2,\cdots,K\}$ 成立,而 σ_1 为恒等置换,所以 $\sigma_{K+m+1}^{-1} = \pi^{-1}$,即 $\sigma_{K+m+1} = \pi$,所以 F 可 K 着色当且仅当 $\pi \in P$.

现在考虑命题的另一个方向. 假设有一族圆弧 F 和 K 种颜色,m 是描述 F 中的圆弧所用到的最大整数. 不失一般性,可以假设对于任意的点 p,$1 \leqslant p \leqslant m$,恰好有 K 条圆弧覆盖 p. 因为若 p 被多于 K 条圆弧覆盖,则必然不能被 K 着色;而当 p 被少于 K 条圆弧覆盖,则可以向 F 中增加若干条圆弧 $\langle p-1, p\rangle$(当 p 为 1 时,为 $(m,1)$),使得不改变 F 的色数并满足恰好有 K 条圆弧覆盖 p.

给定 F,首先将 F 转换成在点 $1,2,\cdots,K+m$ 上的 F^*. 令 $D_1, D_2, \cdots, D_K$ 为 F 中覆盖 1 的圆弧,然后,我们将每一条圆弧 $\langle a,b\rangle \in F - \{D_1, D_2, \cdots, D_k\}$ 替换为 $\langle K+a, K+b\rangle \in F^*$,再将每一条 $D_i = \langle a,b\rangle$ 替换为 $\langle K+a, i\rangle$ 和 $\langle i, K+b\rangle$,添到 F^* 中. 由于在 F^* 中的 K 着色中,$\langle K+a, i\rangle$ 和 $\langle i, K+b\rangle$ 必然有相同的颜色,所以 F^* 可以 K 着色当且仅当 F 可以 K 着色.

注意,这里的 F^* 和前半部分构造的 F 有相同的结构,所以只需要将证明倒着写一遍. 首先将 F^* 分解为 F_i,$i \in \{1,2,\cdots,K\}$,和 C. C 由那些延拓后包含点 1 的弧构成. F_i 按 F_1, F_2, $\cdots$, F_K 的顺序构造,F_i 的弧是从集合 $R(i)$ 中选出的,$R(i) = F^* - C - \bigcup_{j=1}^{i-1} F_i$. 选择规则如下:$F_i$ 的第一条弧为一条以点 i 为起点的弧. 然后,只要在 $R(i)$ 中有一条起点和刚刚添进 F_i 的弧的终点相同的弧,就将其添入 F_i. 因此,F_i 是由一些不相交的弧构成的,且其中的所有弧的延拓的并集为所有从 $i+1$ 到 P_i 的点. 将 C 中的弧按序排列为 $C_1, C_2, \cdots, C_K$,保证 C_i 的起点和 P_i 的终点相同,其中 P_i 为 F_i 中最后添入的弧. 事实上,每个点都严格地被 F^* 中 K 条弧的延拓覆盖保证了上边的过程都得以进行.

现在,构造 $X_1, X_2, \cdots, X_m$ 和 π 作为 WPPSG 的实例,X_j 包含那些整数 $i \in \{1,2,\cdots,K\}$,使得 F_i 包含一条终点为 $K+j$ 的弧. 置换 π 满足,$\pi(i) = j$ 当且仅当 C_j 的终点为 i.

不难发现,从圆弧图着色到 WPPSG 这种转换也是多项式的. 显然在证明中的第一部分,WPPSG 实例转换的结果恰为 F^*,所以相同的证明可以说明 $\pi \in S_{X_1} \cdot S_{X_2} \cdot \cdots \cdot S_{X_m}$,当且仅当 F^* 可以 K 着色.

至此,命题 12.8 证毕.

下面给出一个 WPPSG 问题的算法.

给定 $X_1, X_2, \cdots, X_m$, $L = S_{X_1} \cdot S_{X_2} \cdot \cdots \cdot S_{X_m}$,计算所有 L 中的元素:首先从 $P_0 = \{e\}$ 开始,$P_{j+1} = \{\pi_1 \cdot \pi_2 \mid \pi_1 \in P_j, \pi_2 \in S_{X_{j+1}}\}$,$j = 0,1,\cdots,m-1$. 则 $P = P_m$,很容易可以检查是否 $\pi \in P$. 显然,两个在 $\{1,2,\cdots,K\}$ 的置换的乘积可以在 $O(K)$ 时间内完成,对于每次乘积,最多可能有 $O((K!)^2)$,总的时间复杂度为 $O(m \cdot f(k))$. 至此,由命题 12.8,我们证明了圆弧图的染色有固定参数算法.

12.5 算法 3SAT 的指数时间改进算法.

输入:一个合取范式,每个子句最多包含 3 个文字.

输出:一个满足赋值或宣布这个公式不可满足.

1. 检查当前赋值是否是满足赋值，若是，则输出这个赋值，算法结束.

2. 任取一个在不满足子句中出现的变元 α_1，检查当前不满足的子句中是否同时有正出现和负出现，若否，当所有不满足子句中都为正出现时，赋该变元为真. 当所有不满足子句中都为负出现时，赋该变元为假. 在当前赋值下化简原公式，递归地检查剩余公式的可满足性. 若是，则任取当前不满足的子句 $C_1=\alpha_1 \vee \alpha_2 \vee \alpha_3$，$C_2=\alpha_1 \vee \alpha_4 \vee \alpha_5$ 其中 α_i 都是文字.

2.1 把 α_1 赋值为真，把 α_4 赋值为真，在当前赋值下化简原公式，递归地检查剩余公式的可满足性.

2.2 把 α_1 赋值为真，把 α_4 赋值为假，把 α_5 赋值为真，在当前赋值下化简原公式，递归地检查剩余公式的可满足性.

2.3 把 α_1 赋值为假，把 α_2 赋值为真，在当前赋值下化简原公式，递归地检查剩余公式的可满足性.

2.4 把 α_1 赋值为假，把 α_2 赋值为假，把 α_3 赋值为真，在当前赋值下化简原公式，递归地检查剩余公式的可满足性.

3. 若上述步骤 2.1～2.4 的递归检查全部失败，则宣布没有满足赋值.

命题 12.9 上述算法在 $O^*(1.7693^n)$时间内正确求解 3SAT.

证 我们先来证明上述算法的正确性. 当算法在第 1 步输出一个赋值时，这个赋值的确是满足赋值，因为算法检查过该赋值是满足赋值. 下面证明当算法宣布没有满足赋值时，也是正确的. 当某一变量 α 在子句中都是正文字时，如果该公式有满足赋值，则将所有的 α 赋值为真一定也得到满足赋值；反之亦然. 如果输入公式中含有子句 $C_1=\alpha_1 \vee \alpha_2 \vee \alpha_3$，$C_2=\alpha_1 \vee \alpha_4 \vee \alpha_5$，则所有满足赋值被划分为 4 类.

第一类：把 α_1 赋值为真，把 α_4 赋值为真；

第二类：把 α_1 赋值为真，把 α_4 赋值为假，把 α_5 赋值为真；

第三类：把 α_1 赋值为假，把 α_2 赋值为真；

第四类：把 α_1 赋值为假，把 α_2 赋值为假，把 α_3 赋值为真.

任何一个满足赋值必定属于且仅属于上述一类. 算法在第 2 步递归地检查所有的情况，当步骤 2.1～2.4 所有递归检查都失败时，也就是说上述四类赋值中都没有满足赋值时，输入公式必定没有满足赋值，因为任何一个满足赋值必定属于上述四类之一，所以这时算法在第 3 步宣布没有满足赋值，就是正确的.

由于每次递归至少为一个变量赋值，因此每个递归检查的深度至多为 n，即算法不存在死循环. 这样就证明了算法是正确的.

下面分析算法的复杂性. 设 $T(n)$表示算法在 n 个变量的公式上的运行时间，设公式中有 m 个子句，则 $m=O(n^3)$，因为每个子句至多含有 3 个文字，不同子句的个数至多为 $O(n^3)$. 关于 $T(n)$，与定理 12.2 证明中类似，有下列递推公式

$$T(n) \leqslant 2T(n-2)+2T(n-3)+O(n+m)$$

按照主教材第 1 章中介绍的递推公式的解法，这个递推公式的解是 $T(n)=O^*(1.7693^n)$，其中 1.7693 是方程 $\lambda^3=2\lambda+2$ 的最大根.

12.6 算法如下：

输入：n 个变量的 k-CNF 公式.

输出：该公式的一组满足赋值或宣布没有满足赋值.

下面是该算法的内层循环.

1. 随机均匀地选取初始赋值 $a\in\{0,1\}^n$.

2. 重复 $3n$ 次.

2.1 如果在当前赋值下公式满足,则停止并输出满足赋值;

2.2 找到某个 C 是不可满足的子句;

2.3 显然 C 中不超过 k 个文字,随机选择其中的一个,改变其赋值.

假设该公式可满足,有一个满足赋值 a^*. 下面估计一下上述算法找到 a^* 的概率 p. 一旦知道了 p,显然,上述算法被重复的期望数为 $1/p$. 重复 t 次后,仍得不到满足赋值的概率至多为 $(1-p)^t\leqslant e^{-pt}$. 因此,该算法的复杂性为 $\mathrm{poly}(n)1/p$.

现在计算 p. 定义随机变量 X 为 a 和 a^* 赋值不相同的变量个数. 显然,X 服从二项分布 $B(n,1/2)$,也就是说,$\Pr(X=j)=\binom{n}{j}2^{-n}$,对任意的 $j=0,1,\cdots,n$ 都成立. 当 $X=j$ 时,要想从 a 变成 a^*,需要改变赋值的变元个数为 j. 不妨将赋值的改变看成马尔科夫链,有一个开始状态,有状态 $0,1,\cdots,n$,表示 a 和 a^* 的 Hamming 距离. 从开始状态到 $0,1,\cdots,n$ 的转变代表随机选取初始赋值,从开始状态到状态 j 的概率为 $\binom{n}{j}2^{-n}$. 如果当前处于状态 0,意味着已经找到该满足赋值,该随机过程相应结束. 注意,如果公式有超过 1 个的满足赋值,算法本身可能找到另一个满足赋值,这意味着该随机过程没有在 0 结束,然而这种情况只增加了找到满足赋值的概率.

设 C 是在当前的赋值 a 下的不可满足的子句,那么在 C 中的至多 k 个变量中,至少有一个的赋值与 a^* 不同. 所以,随机改变 C 中的一个变量的赋值,意味着从状态 j 到状态 $j-1$ 的概率至少为 $1/k$,到状态 $j+1$ 的概率至多为 $(k-1)/k$.

假设该随机过程开始转移到状态 j,下面计算该算法到达状态 0 的概率 q_j. 在当前假设下,至少需要 j 步才能到达状态 0,仅考虑哪些走错 i 步的路径,其中 $i\leqslant j$,当走错 i 步时,从状态 j 到状态 0 至少需要 $j+2i$ 步,而 $j+2i\leqslant n+2n=3n$. 根据**投票定理**(the ballot theorem),该随机过程从状态 j 到状态 0 且只走错 i 步的路径数为 $\binom{j+2i}{i}\cdot\frac{j}{j+2i}$. 因此,

$$q_j\geqslant\sum_{i=0}^{j}\binom{j+2i}{i}\cdot\frac{j}{j+2i}\cdot\left(\frac{k-1}{k}\right)^i\cdot\left(\frac{1}{k}\right)^{i+j}\geqslant\frac{1}{3}\sum_{i=0}^{j}\binom{j+2i}{i}\cdot\left(\frac{k-1}{k}\right)^i\cdot\left(\frac{1}{k}\right)^{i+j}$$

利用如下近似公式:

$$\binom{n}{\alpha n}\sim 2^{h(\alpha)n}=\left(\frac{1}{\alpha}\right)^{\alpha n}\left(\frac{1}{1-\alpha}\right)^{(1-\alpha)n}$$

其中,$h(\alpha)=-\alpha\log_2\alpha-(1-\alpha)\log_2(1-\alpha)$ 为二元的熵函数. 特别地,

$$\binom{(1+2\alpha)j}{\alpha j}\text{与}\left[\left(\frac{1+2\alpha}{\alpha}\right)^{\alpha}\cdot\left(\frac{1+2\alpha}{1+\alpha}\right)^{1+\alpha}\right]^j$$

只相差一个多项式的因子. 通过令 $\alpha=\frac{1}{k-2}$,得到 q_j 的下界

$$q_j\geqslant\frac{1}{3}\sum_{i=0}^{j}\binom{j+2i}{i}\cdot\left(\frac{k-1}{k}\right)^i\cdot\left(\frac{1}{k}\right)^{i+j}$$

$$\geqslant\left[\left(\frac{1+2\alpha}{\alpha}\right)^{\alpha}\left(\frac{1+2\alpha}{1+\alpha}\right)^{1+\alpha}\left(\frac{k-1}{k}\right)^{\alpha}\cdot\left(\frac{1}{k}\right)^{1+\alpha}\right]^{j}$$
$$=\left(\frac{1}{k-1}\right)^{j}$$

最后一个不等式是在多项式因子意义下成立的. 利用二项式定理,有如下的关于 p 的估计:

$$p\geqslant\left(\frac{1}{2}\right)^{n}\sum_{j=0}^{n}\binom{n}{j}\left(\frac{1}{k-1}\right)^{j}=\left(\frac{1}{2}\left(1+\frac{1}{k-1}\right)\right)^{n}$$

因此,k-SAT 的复杂度至多为 $\mathrm{poly}(n)\cdot\left(2\left(1-\frac{1}{k}\right)\right)^{n}$.

12.7 在本题中,我们介绍一个确定的局部搜索算法. 注意,在 12.6 题中的随机游动中,在平均情形下,选择了很多初始赋值;对每个初始赋值,开始至多 $3n$ 步的局部搜索. 本题的算法可以看作是 12.6 题算法的去随机,这个去随机自然也分为两部分:初始赋值的去随机和局部搜索的去随机.

为了使描述更加简洁,首先给出一些定义. 定义 Hamming 空间为 $H_n=\{0,1\}^n$,指 n 变量公式的所有赋值集合,Hamming 距离为两个赋值中赋值不同的变量的个数. 定义以 a 为球心,半径为 r 的球为所有与 a Hamming 距离不超过 r 的赋值的集合. 编码集定义为 H_n 的子集. 编码集 M 的覆盖半径定义为

$$r=\max_{u\in H_n}\min_{v\in M}d(u,v)$$

其中,$d(u,v)$为赋值 u、v 的 Hamming 距离. 归一覆盖半径定义为 $\rho=r/n$. 一个半径为 r 的编码集指的是对任意 H_n 的赋值 u,至少存在着一个编码集中的元素 v,使得 $d(u,v)\leqslant r$.

假设得到一个初始赋值 $a\in H_n$,考虑以 a 为球心,半径为 r 的球,所有在这个球中的赋值个数为

$$V(n,r)=\sum_{i=0}^{r}\binom{n}{i}$$

如果归一化覆盖半径 ρ 满足 $0<\rho\leqslant1/2$,那么 $V(n,r)$可以这样估计:

$$\frac{1}{\sqrt{8n\rho(1-\rho)}}\cdot 2^{h(\rho)n}\leqslant V(n,r)\leqslant 2^{h(\rho)n}\qquad(12.1)$$

$h(\rho)=-\rho\log_2\rho-(1-\rho)\log_2(1-\rho)$为二元的熵函数,注意,当 ρ 为常数时,$V(n,r)$与 $2^{h(\rho)n}$ 至多差多项式倍.

在上述的定义下,解的结构有如下的简单的命题.

命题 12.10 F 是需要求解的公式,a 是一个 F 的成假赋值,设 C 是任意的在赋值 a 下为假的子句,则 F 有一个满足赋值在以 a 为球心,以 r 为半径的球中当且仅当 C 中存在一个文字 l,当将 l 赋值为 1 时,把 F 中的 l 都替换为 1 化简后得到的公式记为 $F_{|l=1}$. $F_{|l=1}$有一个满足赋值在以 a 为球心,$r-1$ 为半径的球中.

下面的 Search(F,a,r)是算法中的局部搜索部分,利用深度优先策略,当存在一个满足赋值在以 a 为球心,以 r 为半径的球中,返回真(True),否则返回假(False).

程序:Search(F,a,r)

1. 在当前赋值 a 下,如果 F 的所有子句为真,则返回真.

2. 如果 $r\leqslant0$,返回假.

3. 如果 F 包含空子句,返回假.

4. 根据某个确定性规则,选择一个在赋值 a 下成假子句 C,对每一个 C 中的文字 l,进行 Search($F_{|l=1}$, a, $r-1$). 如果存在某个 l,Search($F_{|l=1}$, a, $r-1$)返回真,则返回真;否则返回假.

由命题12.10,如果 F 在 a 为球心,r 为半径的球中有满足赋值,上述算法总会找到的.

命题 12.11 Search(F, a, r)的时间复杂度为 $\mathrm{poly}(n)\cdot k^r$.

证 Search(F, a, r)的递归深度至多为 r,由于 F 的每个子句至多有 k 个文字,所以每次递归调用不会超过 k 个,即递归树的叶子不会超过 k^r 个.

注意,k^r 可以远远小于 $V(n,r)$,意味着可以得到改进的指数时间算法.

命题 12.12 对任意 H_n,$n\geqslant 1$,存在半径为 r 的编码集 M,并且 $|M|\leqslant n\cdot 2^n/V(n,r)$.

证 (概率方法)随机均匀有重复地从 H_n 中选择 $n\cdot 2^n/V(n,r)$ 个赋值,我们证明这些赋值以大概率形成一个半径至多为 r 的编码集. 设 a 为 H_n 的一个赋值,对另一随机赋值 b,它属于 b 为球心,半径为 r 的球的概率为 $V(n,r)/2^n$,a 不属于任意随机赋值为球心,半径为 r 的球的概率为

$$(1-V(n,r)/2^n)^{n\cdot 2^n/V(n,r)}\leqslant \mathrm{e}^{-n}$$

利用任意 x,$1+x\leqslant \mathrm{e}^x$. 利用并的界,随机选择的元素构成一半径为 r 的代表集的概率至少为 $1-2^n\mathrm{e}^{-n}$,当 n 趋向于 ∞ 时趋向于1.

命题 12.13 设 $0<\rho<1/2$,对任意的 n,存在编码集 M,使得 M 半径至多为 ρn,$|M|$ 至多为 $n\sqrt{n\rho(1-\rho)}2^{(1-h(\rho))n}$.

证 直接利用命题12.12和(12.1)式可得.

命题12.13说明了存在几乎最优的编码集,命题12.12提供了一个随机算法得到这个编码集,下面逐步得到一个确定的算法,来得到一个并不太坏的编码集.

编码集算法. 我们将这个编码集问题看成集合覆盖问题:即用最少的半径为 ρn 的球去覆盖 H_n,用如下的贪心法,它和最优解至多相差 n 倍. 贪心策略是:在每一步,选择一个覆盖没有被覆盖赋值最多的球.

我们来考虑一下上述算法的时间复杂度,对每一步,选择覆盖未覆盖赋值最多的球,并更新已覆盖赋值,至多用 $\mathrm{poly}(n)\cdot 2^{2n}$,而至多有 2^n 步循环,因此有如下的命题:

命题 12.14 设 $n\geqslant 1$,$0<\rho<1/2$,$\beta(n)=\sqrt{n\rho(1-\rho)}$. 那么一个半径至多为 ρn,大小至多为 $n^2\beta(n)2^{(1-h(\rho))n}$ 的编码集可以在 $\mathrm{poly}(n)\cdot 2^{3n}$ 时间内得到.

显然,命题12.14的时间是不可忍受的,下面对编码集中每个编码的长度做出一些限制,以使得可以在更少的时间内得到编码集.

命题 12.15 设 $d\geqslant 2$,d 是 n 的一个因子,$n\geqslant 1$,$0<\rho<1/2$. 则存在一个多项式 q_d,一个半径至多 ρn 和大小至多为 $q_d(n)\cdot 2^{(1-h(\rho))n}$ 的编码集 M 可以在时间 $q_d(n)\cdot(2^{3n/d}+2^{(1-h(\rho))n})$ 内得到.

证 将 H_n 的赋值看作一个长为 n 的位串,因为 d 是 n 的一个因子,可以将 H_n 中的每个赋值都分成 d 段,每段长度为 n/d. 根据命题12.14,可以构造 $H_{n/d}$ 的编码集 M',并使得 M' 的覆盖半径至多为 $\rho n/d$.

令 M 为 M' 中所有的 d 个串的连接组成的新串的集合. 显然,M 的覆盖半径不会超过

ρn,因为对于每一个 $a\in H_n$,可以将其分成 d 段,$a_1,a_2,\cdots,a_d$,每一段长为 n/d. 构造 M'的时间为 $O(q(n)\cdot 2^{3n/d})$,M 的大小至多为 $(n^2\beta(n)2^{(1-h(\rho))n/d})^d=n^{2d}\beta^d(n)2^{(1-h(\rho))n}$.

k-SAT 的确定的局部搜索算法如下：

输入：k-CNF 公式 F,共有 n 个变量.

输出：若 F 有满足赋值,返回真;否则返回假.

1. 令 $\rho=\dfrac{1}{k+1}$.

2. 利用命题 12.15,令 $d=6$,产生一个编码集 M,半径至多为 ρn.

3. 对 M 中的每一个赋值,调用 Search$(F,a,\rho n)$. 当至少有一个子程序返回真时返回真,所有子程序返回假时返回假.

下面对该算法的复杂度进行分析. 首先,根据命题 12.15,用半径为 ρn 的球覆盖 H_n,这个覆盖包含 $B(n,\rho,d)=q_d(n)\cdot 2^{(1-h(\rho))n}$ 个球,它的构造的复杂度 $T_1(n,\rho,d)=q_d(n)\cdot(2^{3n/d}+2^{(1-h(\rho))n})$,其中 $q_d(n)$是一个多项式. 在每个球内,局部搜索所用时间为 $T_2(n,\rho)=p(n)\cdot k^{\rho n}$,$p$ 也是多项式. 所以,该算法运行的总时间复杂度为

$$
\begin{aligned}
&T_1(n,\rho,d)+B(n,\rho,d)\cdot T_2(n,\rho)\\
&=q_d(n)\cdot(2^{3n/d}+2^{(1-h(\rho))n})+q_d(n)\cdot 2^{(1-h(\rho))n}\cdot p(n)\cdot k^{\rho n}\\
&=\mathrm{poly}(n)\cdot 2^{n\left(1+\frac{1}{k+1}\log_2\frac{1}{k+1}+\frac{k}{k+1}\log_2\frac{k}{k+1}+\frac{1}{k+1}\log_2 k\right)}\\
&=\mathrm{poly}(n)\cdot 2^{n\left(1+\log_2\frac{k}{k+1}\right)}\\
&=\mathrm{poly}(n)\cdot\left(2-\frac{2}{k+1}\right)^n
\end{aligned}
$$

12.8 (示例)用模拟退火法计算 $f(x)=x^2-5x+29$ 的极小值. 算法如下：

1. 生成一个初始解 $x_0=0$,设定初始温度 $T_0=100\,000$.

2. 循环执行下列步骤,直到 $T<0.000\,000\,01$.

2.1 从当前解 x 出发,随机增大或缩小 0.25.

2.2 若新解是改进解,则代替当前解;否则新解的退步幅度 Δ 定义为新解与旧解的函数值之差,以概率 $e^{-\frac{\Delta}{T}}$用新解代替当前解.

2.3 更新温度 $T=0.9T$.

3. 输出见到的最好的解.

在实验中,该温度以几何级数递减,用 200 多步就可以收敛至最优解. C++ 代码如下：

```
#include<iostream>
#include<math.h>
#include<time.h>
using namespace std;
double f(double x) {
    return x * x-5 * x+29;
}
int main() {
    double x=0,oldx=0;
    double T=100000;
```

```
    double bestx=x;
    srand((unsigned) time(NULL));
    while(T>=0.00000001) {
        int random=rand();
        x=oldx+0.5 * ((random %2)-0.5);
        if(f(x)<f(oldx)) {
            oldx=x;
            if(f(x)<f(bestx))
                bestx=x;
        } else {
            double p=pow(2.718281828,-(f(x)-f(oldx))/T);
            double randomEvent=(rand() %10000)/10000.0;
            if(randomEvent<p) {
                oldx=x;
            }
        }
        T=T * 0.9;
    }
    cout<<bestx<<endl;
}
```

12.9 对每个顶点 v,其余 $n-1$ 个顶点与 v 是否有边相连是独立的,且服从 0-1 分布,且有边的概率为 1/2,设 X_i 为第 i 个顶点与 v 相连的边数,$i=1,2,\cdots,n-1$.

性质一的证明:根据切诺夫界,

$$\begin{aligned}\Pr\left[\left|\sum_{i=1}^{n-1}X_i-\frac{n}{2}\right|\geqslant\frac{n}{50}\right]&\leqslant\Pr\left[\left|\sum_{i=1}^{n-1}X_i-\frac{n-1}{2}\right|\geqslant\frac{n-1}{100}\right]\quad(\text{当 } n \text{ 充分大时})\\&=\Pr\left[\left|\sum_{i=1}^{n-1}X_i-\mathrm{E}\left[\sum_{i=1}^{n-1}X_i\right]\right|\geqslant\frac{1}{50}\mathrm{E}\left[\sum_{i=1}^{n-1}X_i\right]\right]\\&\leqslant 2\exp\left(-\min\left\{\frac{4}{50^2},\frac{1}{100}\right\}\mathrm{E}\left[\sum_{i=1}^{n-1}X_i\right]\right)\\&=2\exp\left\{-\frac{4}{2500}\times\frac{n-1}{2}\right\}\xrightarrow{n\to\infty}0\end{aligned}$$

上边只证明了对于固定的顶点 v,满足性质一.

$$\begin{aligned}\Pr[\forall v,v\text{ 满足性质一}]&=1-\Pr[\exists v,v\text{ 不满足性质一}]\quad(\text{并的界})\\&\geqslant 1-n\Pr[v\text{ 不满足性质一}]\xrightarrow{n\to\infty}1\end{aligned}$$

所以,对每个顶点 v,$|N(v)|$ 介于 $n/2$ $n/50$ 之间.

性质二的证明:对每对顶点 u 和 v,$N(u)\cup N(v)$ 是 u 和 v 邻域的并,由于在 $G(n,p)$ 模型中,每条边是否出现是独立的. 所以对第三个点 w,w 是否属于 $N(u)\cup N(v)$ 是独立同分布的 0-1 随机变量,且属于的概率为 3/4,类似性质一的证明,容易得到性质二的证明.

性质三的证明:对 $V\backslash\{u,v,w\}$ 中的任一顶点,是否属于 $N(u)\cup N(v)\cup N(w)$ 是独立同分布的,且属于的概率为 7/8,类似 1 的计算,可得证.

下面根据上述三条性质，对如下算法进行分析.

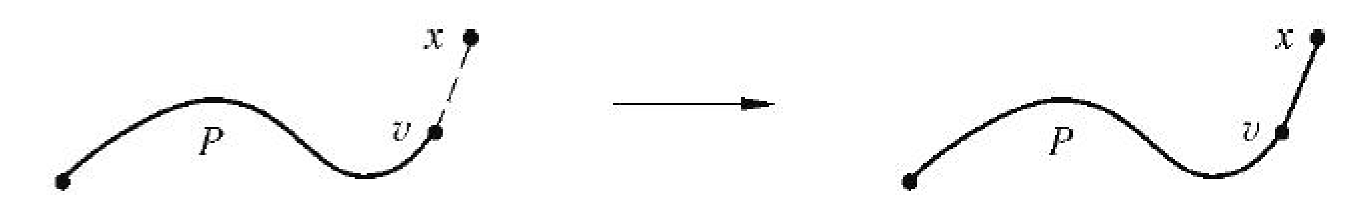
算法 哈密顿回路问题在 $G(n,1/2)$ 上的有效算法.

输入：在 $G(n,1/2)$ 分布下产生的一个随机图 G.

输出：G 中的一条哈密顿回路.

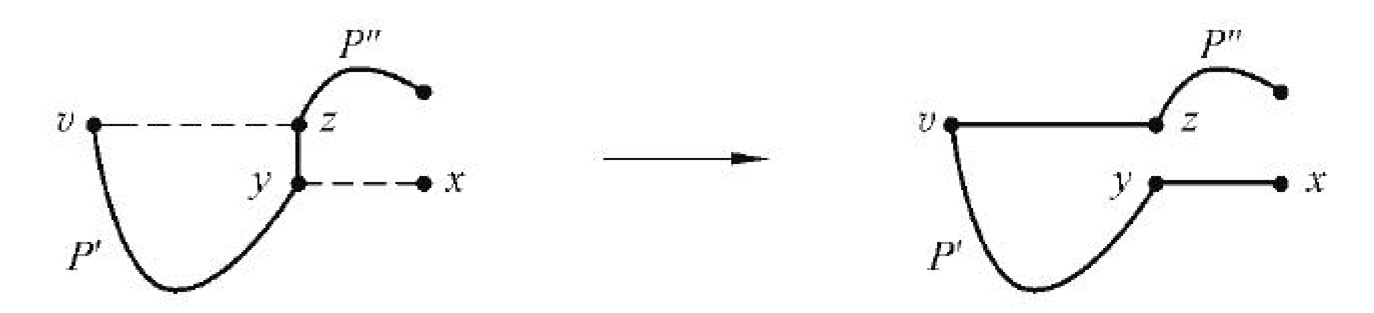
1. 反复利用下面两条规则构造一条路径 P，直到 P 不能再延长为止.

1.1 若 P 外的顶点 x 与 P 的端点 v 相邻，则用边 (x,v) 把 x 加入 P，即令 $P=xP$.



分析：这时，如果利用 1.1 已经不能延长该路径，则说明对 P 外的顶点 x，x 不与 P 的两端相邻，按照性质二，这时，以大概率，路径上至少已经有 $3n/4-n/50$ 个顶点.

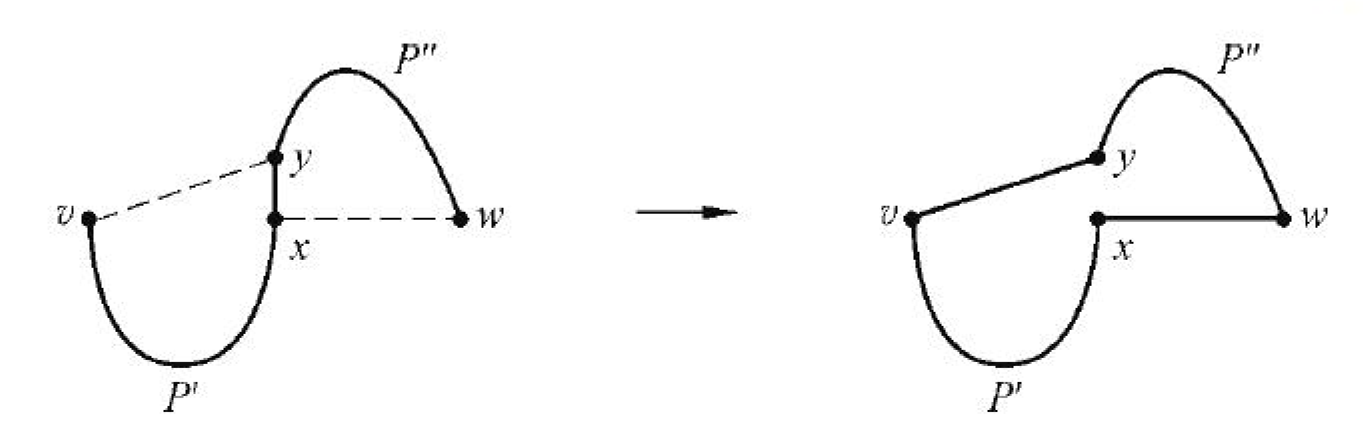
1.2 若 $P=vP'yzP''$ 且 v 与 z 相邻、y 与 P 外的顶点 x 相邻，则令 $P=xyP'vzP''$.



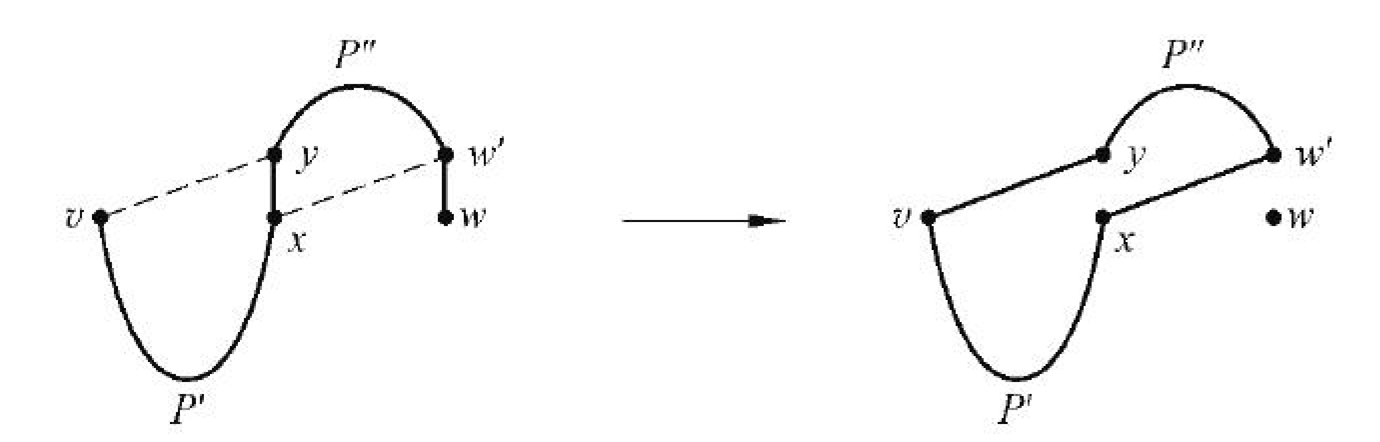
分析：假设已经没有符合 1.1 的 x，如上图，按照性质一，至少有 $n/2-n/50$ 个顶点与 v 相邻，这些点不能在已有路径外，也就是说，以大概率，可以找到如图的 z，所以不能加入路径上的点也不能与 y 相邻，所以步骤 1.2 结束后，路径外的点至少不能和该路径上的三个点相邻，按照性质三，路径上至少有 $7n/8-n/50$ 个顶点.

2. 反复利用下面两条规则构造一个圈 C.

2.1 若 $P=vP'xyP''w$ 且 v 与 y 相邻、w 与 x 相邻，则令 $C=wxP'vyP''w$.



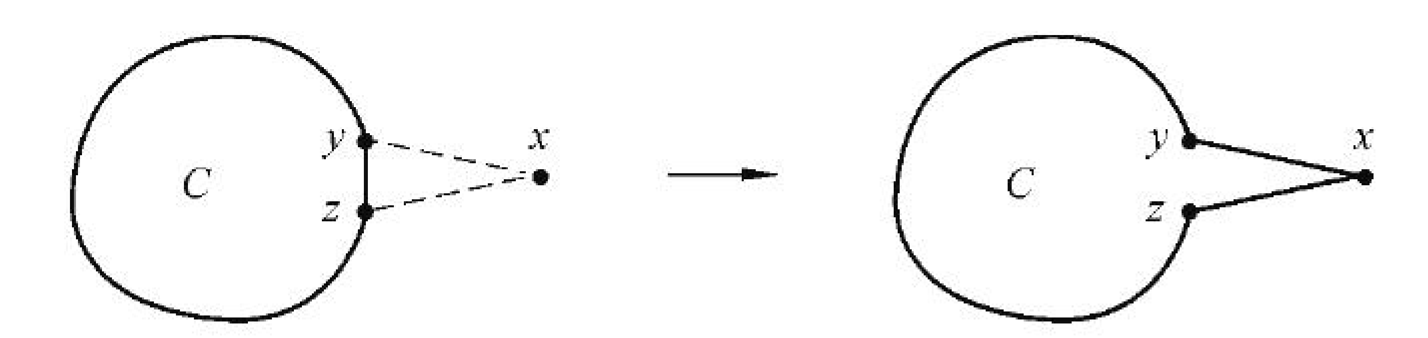
2.2 若 $P=vP'xyP''w'w$ 且 v 与 y 相邻、w' 与 x 相邻，则令 $C=w'xP'vyP''w'$.



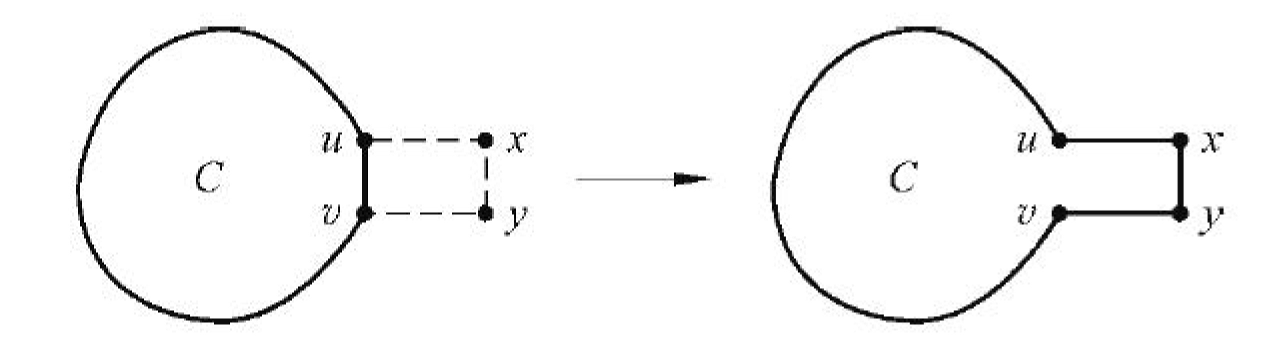
分析：以大概率，v 和 w 都有 $n/2-n/50$ 个相邻顶点在该路径中，如上图，假设 v 与 y 相邻，则相应的 y 有 $n/2-n/50$ 个，因为 x 可能就是 v，所以相应的 x 也至少 $n/2-n/50-1$ 个，而按照性质二，w 和 w' 的邻域至少有 $3n/4-n/50$ 个顶点，由于 $3n/4-n/50>n-(n/2-n/50-1)$，所以必然会构成一个圈 C，且 C 上最多比第一步构造的路径少一个顶点，即至少有 $7n/8-n/50-1$ 个顶点.

3. 反复利用下面三条规则把其余顶点加入 C,每次加入一个或两个顶点,直到没有剩余顶点为止.

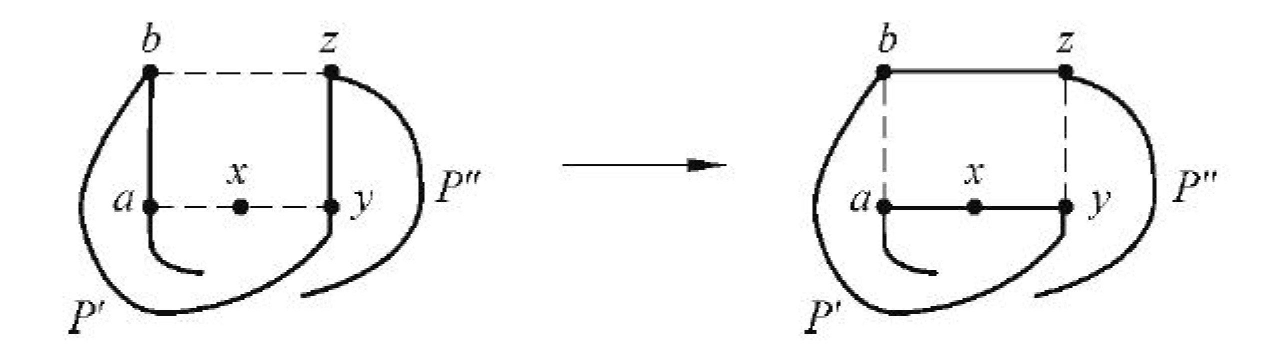
3.1 若 C 外的顶点 x 与 C 上连续两点 y 和 z 都相邻,则令 $C=C\cup\{(y,x),(x,z)\}-\{(y,z)\}$.



3.2 若 C 外的两点 x 和 y 相邻且分别与 C 上连续两点 u 和 v 相邻,则令 $C=C\cup\{(u,x),(x,y),(y,v)\}-\{(u,v)\}$.



3.3 若 $C=abP'yzP''a$ 且 C 外的顶点 x 与 a 和 y 都相邻、b 与 z 相邻,则令 $C=axyP'bzP''a$.



分析:首先,只要有边存在,如步骤3.2中的 x 和 y,则证明至少能用步骤3.1或步骤3.2中的一种方法. 下面假设两个方法都不能用. 由于与 x 和 y 之一相邻的点至少有 $3n/4-n/50$ 个,而这些点在 C 上至少有 $3n/4-n/50-(n-(7n/8-n/50-1))=5n/8-2n/50-1>n/2$,它们不可能出现在一个至多为 n 个顶点的圈,且两两不相邻. 假设圈外的顶点是一个独立集,且不能用3.1,为了分析方便,我们利用 $G(n,1/2)$ 的独立集为 $O(n)$ 量级这一事实. 而相应的与 x 相邻的顶点至少有 $n/2-n/50$ 个,规定圈上的一个方向,则这些的顶点同侧的顶点也有 $n/2-n/50$ 个(因为这些与 x 相邻的顶点在圈上互不相邻),所以它们必然存在着一条边,也就是如图中的 bz 边.

4. 输出 C 作为哈密顿回路.

算法复杂性分析:由于我们可以用一个标记数组,所以对于一个顶点,其是否在已有的路径或圈中可以在常数时间内得知. 所以,在第1步中,每加入一个顶点,可以在 $O(n)$ 时间内完成,顶点数至多为 n,故在有顶点加入时,复杂性不会超过 $O(n^2)$;而判断是否还有顶点可以加入,显然只需将所有顶点遍历一遍,对于每一个顶点,也可在 $O(n)$ 时间内判断是否可以用两种方法加入路径. 第2步只涉及 $O(1)$ 个顶点,所以可以在 $O(1)$ 时间内完成. 在第3步中,如果是利用步骤3.1或步骤3.2,则每添加一个顶点,最多将图中顶点遍历一遍以及访问 $O(n)$ 条边,故这一部分时间复杂度不会超过 $O(n^2)$;对于步骤3.3,要更进一步地分析.

设在圈上有与 x 相邻的按顺时针排列的两个不相邻的顶点 a 和 b，将沿顺时针方向排列的下一个顶点简称为后边的顶点，设 a 后边为 u，b 后边为 v，假设此时 x 所有邻域中的顶点都在圈上（否则可以利用步骤 3.1 或步骤 3.2 将 x 添入），则设 x 的邻域为 $A=\{a,b,s_1,s_2,\cdots\}$，设这些顶点后边的顶点集为 $B=\{u,v,t_1,t_2,\cdots\}$，则与 u、v 之一相邻的顶点至少有 $3n/4-n/50$ 个，而 B 中的顶点至少有 $n/2-n/50$ 个，$3n/4-n/50+n/2-n/50>n$，所以 B 中有和 u 或 v 相邻的顶点. 所以步骤 3.3 的添加可以首先找到两个与 x 相邻顶点 a 和 b，这需要 $O(1)$ 的时间，然后遍历一遍圈，肯定有和 u 或 v 相邻的顶点，这时可以按步骤 3.3 将 x 加入圈，而这遍历需要 $O(n)$ 的时间，所以总的时间复杂度为 $O(n^2)$.

12.10 一些已经生成的算例可在以下网址找到（包括目前的求解记录）：

http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/graph-benchmarks.htm.

12.11 一些已经生成的算例可在以下网址找到：

http://www.nlsde.buaa.edu.cn/~kexu/benchmarks/benchmarks.htm.

12.12 由定义，$II^\dagger=I$，$XX^\dagger=I$，$ZZ^\dagger=I$，$HH^\dagger=I$.

$\text{CN CN}'=\begin{pmatrix}I & \\ & X\end{pmatrix}\begin{pmatrix}I & \\ & X'\end{pmatrix}=\begin{pmatrix}I & \\ & I\end{pmatrix}$，按照酉变换的定义，以上量子门都是酉变换.

12.13 输入：$|0\rangle|0\rangle\cdots|0\rangle|1\rangle$（即 n 个 $|0\rangle$，1 个 $|1\rangle$），以及函数 $f:\{0,1\}^n\to\{0,1\}$.

输出：判断 f 在所有输入取相同值，或者在一般输入上取 0，另一半输入上取 1.

算法步骤：

1. 作用哈达玛门到每个比特，得到

$$\frac{1}{\sqrt{2^{n+1}}}(|0\rangle+|1\rangle)^n(|0\rangle-|1\rangle)=\frac{1}{\sqrt{2^{n+1}}}\sum_{x=0}^{2^n-1}|x\rangle(|0\rangle-|1\rangle)$$

其中，x 为二进制数.

2. 用 f 作用，得

$$\frac{1}{\sqrt{2^{n+1}}}\sum_{x=0}^{2^n-1}|x\rangle(|f(x)\rangle-|1\oplus f(x)\rangle)$$
$$=\frac{1}{\sqrt{2^{n+1}}}\sum_{x=0}^{2^n-1}(-1)^{f(x)}|x\rangle(|0\rangle-|1\rangle)$$

3. 对每一比特用哈达玛门作用，于是

$$H\left(\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)\right)=|1\rangle$$

设 $H(|x\rangle)=H(|x_{n-1}x_{n-2}\cdots x_0\rangle)=\frac{1}{\sqrt{2^n}}\sum_{y=0}^{2^n-1}a_y|y\rangle$，其中 $|y\rangle=|y_{n-1}y_{n-2}\cdots y_0\rangle$，注意

$$a_y=\prod_{k=0}^{n-1}(-1)^{x_k y_k}$$

故第 3 步结束后的运算结果为

$$\left(\frac{1}{2^n}\sum_{x=0}^{2^n-1}(-1)^{f(x)}\sum_{y=0}^{2^n-1}(-1)^{x_0y_0}\cdots(-1)^{x_{n-1}y_{n-1}}|y\rangle\right)|1\rangle$$

4. 测量第3步的运算结果,因为测量到$|0\rangle|0\rangle\cdots|0\rangle|1\rangle$的概率为

$$\left|\frac{1}{2^n}\sum_{x=0}^{2^n-1}(-1)^{f(x)}\right|^2$$

所以如果f为常值,则以概率1得到$|0\rangle|0\rangle\cdots|0\rangle|1\rangle$,否则若$f$一半取0,一半取1,则不会测量到$|0\rangle|0\rangle\cdots|0\rangle|1\rangle$.

参 考 文 献

［1］ 屈婉玲，刘田，张立昂，王捍贫．算法设计与分析．北京：清华大学出版社，2011.

［2］ 耿素云，屈婉玲，王捍贫．离散数学教程．北京：北京大学出版社，2002.

［3］ Jon Kleinberg，Eva Tardos. 算法设计．张立昂，屈婉玲译．北京：清华大学出版社，2007.

［4］ Thomas H Cormen，Charles E Leiserson，Ronald L Rivest. Introduction to Algorithms (Second Edition). The MIT Press，2001. 北京：高等教育出版社，2002.

［5］ Hu T C. Combinatorial Algorithms. Addison-Wesley Publishing Company，1982.

［6］ Alfred V Aho，John E Hopcroft，Jeffrey D Ullman. The Design and Analysis of Computer Algorithms. 北京：机械工业出版社，2006.

［7］ Sanjoy Dasgupta，Christos Papadimitriou，Umesh Vazirani. Algorithms. The McGraw-Hill Companies，2008.

［8］ 张立昂．可计算性与计算复杂性导引(第 3 版)．北京：北京大学出版社，2011.

［9］ Papdimitriou C H，K Steiglitz. Combinatorial Optimization，Algorithms and Complexity. Printice-Hall Inc. 1982. 组合最优化——算法和复杂性．刘振宏，蔡茂诚译．北京：清华大学出版社，1988.

［10］ 王晓东．算法设计与分析(第 2 版)．北京：清华大学出版社，2008.

［11］ Alan Frieze，Bruce Reed. Probabilistic Analysis of Algorithms. In：Probabilistic Methods for Algorithmic Discrete Mathematics，Algorithms and Combinatorics Vol. 16，36-92，1998.

［12］ Ke Xu，Wei Li. Exact Phase Transitions. In：Random Constraint Satisfaction Problems. Journal of Artificial Intelligence Research，12：93-103，2000.

［13］ Ke Xu，Wei Li. Many Hard Examples in Exact Phase Transitions. Theoretical Computer Science，355(3)：291-302，2006.

［14］ Ke Xu，F Boussemart，F Hemery and C Lecoutre. Random Constraint Satisfaction：Easy Generation of Hard(Statisfiable) Instances. Artificial Intelligence，171：514-534，2007.

［15］ Michael Nielsen，Isaac Chuang. Quantum Computation and Quantum Information. 北京：高等教育出版社，2003.

［16］ Gerhard Woeginger. Exact Algorithms for NP-Hard Problems：A Survey，In：Combinatorial Optimization(Edmonds Festschrift). Lecture Notes in Computer Science vol. 2570，185-207，2003.

［17］ Abraham Flaxman. Algorithms for Random 3-SAT，In：The Encyclopedia of Algorithms，742-744，2008. Extended Version：http://www. math. cmu. edu/～adf/research/rand-sat-algs. pdf

［18］ Sanjeev Arora，Boaz Barak. Computational Complexity：A Modern Approach. Cambridge University Press，2009.

［19］ Avrim Blum. Lecture 3-Probabilistic Analysis and Randomized Quicksort，Lecture notes in an Algorithms course. Carnegie Mellon University，2009. http://www. cs. cmu. edu/～avrim/451f09/lectures/lect0901. pdf

［20］ Wing-Kai Hon(韩永楷). Lecture 21-Markov Chains(Definition，Solving 2SAT). Lecture slides in an Randomized Algorithms course，National Tsinghua University，2009. http://www. cs. nthu. edu. tw/～wkhon/random09/lecture/lecture21. pdf